USENIX

Proceedings of the 22nd Large Installation System Administration Conference

San Diego, CA, USA November 9–14, 2008

conference

proceedings

22nd Large Installation System Administration Conference

San Diego, CA, USA

November 9–14, 2008

Sponsored by
USENIX and SAGE

USENIX®
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

THE USENIX SIG FOR
[sage]
SYSADMINS

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: +1 510-528-8649
http://www.usenix.org
<office@usenix.org>

Past LISA Conferences

For past LISAs, see http://www.usenix.org/events/bytopic/lisa.html.

ISBN-13: 978-1-931971-63-8

Printed in the United States of America on 40% recycled paper with 40% post-consumer waste.

USENIX Association

Proceedings of the 22nd Large Installation System Administration Conference (LISA ’08)

November 9-14, 2008
San Diego, California, USA

ACKNOWLEDGMENTS

PROGRAM CHAIR
Mario Obejas, *Raytheon*

PROGRAM COMMITTEE
Paul Anderson, *University of Edinburgh*
Derek Balling, *Answers Corporation*
Travis Campbell, *AMD*
Narayan Desai, *Argonne National Laboratory*
Æleen Frisch, *Exponential Consulting*
Brent Hoon Kang, *University of North Carolina at Charlotte*
Chris McEniry, *Sony Computer Entertainment America*
Adam Moskowitz, *SiCortex, Inc.*
David Parter, *University of Wisconsin*
David Plonka, *University of Wisconsin*
Kent Skaar, *BMC Software, Inc.*

INVITED TALKS COORDINATORS
Rudi van Drunen, *Competa IT/Xlexit*
Philip Kizer, *Tekelec*

WORKSHOPS COORDINATOR
Lee Damon, *University of Washington*

GURU IS IN COORDINATOR
John "Rowan" Littell, *California College of the Arts*

WORK-IN-PROGRESS REP'TS/POSTERS COORDINATORS
Brent Hoon Kang, *University of North Carolina at Charlotte*
Gautam Singaraju, *University of North Carolina at Charlotte*

SCRIBE
Esther Filderman, *University of Michigan*

FIGHTING SPAM WORKSHOP
Chris St. Pierre, *Nebraska Wesleyan University*

MICROLISA WORKSHOP
Robert Au

BUSINESS-DRIVEN IT MANAGEMENT WORKSHOP
Mark Burgess, *Oslo University College*
Claudio Bartolini, *HP Labs Palo Alto*

PRACTICAL CONFIGURATION MANAGEMENT WORKSHOP
Narayan Desai, *Argonne National Laboratory*

UNIVERSITY ISSUES WORKSHOP
John "Rowan" Littell, *California College of the Arts*
Adele Shakal, *University of Southern California*

ZFS WORKSHOP
Richard Elling, *Sun Microsystems*

GOVERNMENT/MILITARY SYSADMIN WORKSHOP
Andrew Seely, *Science Applications International Corporation*

ADVANCED TOPICS WORKSHOP
Adam Moskowitz, *SiCortex, Inc.*

SERVER ROOM BEST PRACTICES WORKSHOP
Hunter Matthews, *Duke University*

VIRTUAL INFRASTRUCTURES WORKSHOP
Paul Anderson, *University of Edinburgh*

USENIX BOARD LIAISON
Alva Couch, *Tufts University*

PROCEEDINGS TYPESETTING
Rob Kolstad

CONFERENCE ORGANIZATION
The USENIX Association Staff

EXTERNAL REVIEWERS

Paul Anderson
Bob Apthorpe
Derek Balling
Lori Barfield
Matthew Barr
Lois Bennett
Bill Bogstad
Travis Campbell
William Cheswick
Marc Chiarini
Alva Couch
Narayan Desai
Esther Filderman
Jon E. Finke
Scott Francis
Æleen Frisch
David Harnick-Shapiro
Jim Hickstein
Andrew Hume
Brent Hoon Kang
William LeFebvre
Paul Lussier
Chris McEniry
Dustin J. Mitchell
Katerina Mitrokosa
Adam Moskowitz
Mario Obejas
David Parter
David Plonka
Chris St. Pierre
John Sellens
Josh Simon
Kent Skaar
Marc Staveley
John Stoffel
Leon Towns-von Stauber
Pat Wilson

CONTENTS

INDEX OF AUTHORS

Message from the Program Chair

Welcome to LISA ’08!

My first LISA was in 1997. I had switched to sysadmin work from programming about 6 years earlier. I don’t remember exactly how I discovered LISA, but it was most likely through searching USENET (how times have changed) for training reviews. I was able to persuade my employer to fund me for the tutorials but not the tech sessions.

Wandering through the registration hall and taking in the sights and conversations, I realized I had discovered a subculture. When I returned to my wife, she remarked, “Everybody here is dressed like you.” Confirmed – my people!

In the next three days – especially in the BoFs, lunches, and famous hallway track – I reveled in being among people with the same technical and career challenges I had, in places large and small.

Fast-forward to 2008. The LISA Conference has been a priority in my annual training plan ever since. The history of our profession is unfolding here and being recorded in no small part in the written artifacts of this conference.

Make no mistake, putting together this conference takes work. The efforts of the USENIX staff and Organizing Committee have resulted in the excellent set of tutorials, invited talks, workshops, Guru track, WiPs, and posters presented at LISA ’08. Forty-two brave souls submitted papers to our refereed papers track; 17 of those were chosen for further refinement and publication. Especially noteworthy are the timely keynote and plenary session speakers we have in our lineup.

For half of you, this is your first LISA. I encourage you to engage in discussions, ask questions, mingle and make contacts. For the other half, the LISA veterans, I encourage you to similarly stretch out of your comfort zone. Go attend a session on a topic you are not that familiar with. If you are a technical topic junkie, go check out one of those squishy, subjective, non-technical topics you usually avoid. And please reach out to the LISA newbies; they are intimidated enough as it is, especially the ones who came here alone. Encourage them to join discussions, attend a BoF with you, or ask a speaker their question. For all of you who came with a group, please try to do some of the sessions and activities without that group.

If you like what you see and hear this week, please let the organizers know that it worked for you. It’s one of the few explicit thanks they will get. If the speakers, tutorial instructors, paper and poster presenters, Gurus, workshop organizers, and USENIX staff contributed to your week, please tell them so. And please start thinking about what *you* might contribute to LISA ’09.

The reason I tell my employer I want to attend LISA is for the intellectual stimulation, the training, the exposure to the state of my profession, and networking with colleagues. For these and other reasons, this conference has earned a special place in my heart. I hope it will have a similar effect on you.

Mario Obejas
Program Chair
LISA ’08

Designing Tools for System Administrators: An Empirical Test of the Integrated User Satisfaction Model

Nicole F. Velasquez, Suzanne Weisband, and Alexandra Durcikova – University of Arizona

ABSTRACT

System administrators are unique computer users. As power users in complex and high-risk work environments, intuition tells us that they may have requirements of the tools they use that differ from those of regular computer users. This paper presents and empirically validates a model of user satisfaction within the context of system administration that accounts for the needs of system administrators. The data were collected through a survey of 125 system administrators and analyzed using structural data modeling techniques. The empirical results of this preliminary investigation demonstrate that user satisfaction models are appropriate in the context of system administration and support the idea that system administrators have unique system and information needs from the tools they use.

Introduction

System administrators (sysadmins) are becoming increasingly important as organizations continue to embrace technology. With responsibilities that can include the installation, configuration, monitoring, troubleshooting, and maintenance of increasingly complex and mission-critical systems, their work distinguishes them from everyday computer users, and even from other technology professionals. As technology experts and system power users, sysadmins are clearly not novice users; however, most software is designed with novices in mind [Bodker, 1989]. Their broad areas of responsibility often result in a "juggling act" of sorts, quickly moving between tasks, and often not completing a given task from beginning to end in one sitting [Barrett, et al., 2004].

Also differentiating system administrators from regular end users of computer systems is the environment in which they work. As more business is conducted over the Internet, simple two-tier architectures have grown into complex n-tier architectures, involving numerous hardware and software components [Bailey, et al., 2003]. Because this infrastructure must be managed nearly flawlessly, the industry has seen system management costs exceed system component costs [IBM, 2006; Kephart and Chess, 2003; Patterson, et al., 2002]. In addition, any system downtime can result in significant monetary losses. Although many vendors are exploring automated system management to cope with these complex and risky environments [HP, 2007; IBM, 2006; SunMicrosystems, 2006], these tools offer little comfort to system administrators, as the sysadmins are often held responsible for any system failures [Patterson, et al., 2002].

Citing the unique problems they face because of the complex systems they manage, their risky work environment, and their power-user access, authorities and skills, Barrett, et al. [Barrett, et al., 2003] call for a focus on system administrators as unique users within HCI research. By examining the work practices of sysadmins, practitioners can design and develop tools suited to their specific needs. With the human cost of system administration now exceeding total system cost [IBM, 2006], the importance of catering to these specialized users is apparent.

To investigate tool features important to system administrators, we utilized a multi-method approach, including semi-structured interviews and a review of previous system administrator research. Our study participants included both junior and senior system administrators whose work responsibilities included the administration of networks, storage, operating systems, web hosting, and computer security. The system administrators we studied worked in enterprise or university settings. Our observations of and conversations with our participants allowed us to gain a better understanding of how the work is accomplished. Semi-structured interviews gave us the opportunity to ask more pointed questions about the sysadmin's motivations and reasons for their particular work routines and allowed us to collect their opinions on why they choose to use or not use a given tool to accomplish their work. With the insights we gained from these investigations, we turned our efforts to a review of the existing system administrator studies to confirm our findings.

Important Characteristics

The strength of a focused investigation of technology-in-use lies in its ability to identify realistic solutions and guide potential designs [Button and Harper, 1995]. By examining the work of system administrators and reviewing previous studies of system administrators (e.g., [Bailey, et al., 2003; Bailey and Pearson, 1983; Barrett, et al., 2004; Button and Harper, 1995; Fitzpatrick, et al., 1996; Haber and Bailey, 2007; Haber and Kandogan, 2007], we have generated the following list of attributes that appear to be important to system administrators. (The reader should note that many attribute definitions were refined throughout the project, referencing the attribute definitions provided in [Wixom and Todd, 2005].)

Information Attributes	System Attributes
Logging	Flexibility
Accuracy	Scalability
Completeness	Monitoring
Format	Situation Awareness
Currency	Scriptability
Verification	Accessibility
	Integration
	Speed
	Reliability
	Trust

Table 1: Information and system attributes.

1. Flexibility: the way the system adapts to changing demands of the system administrator
2. Scalability: the ability of a system to scale to large and/or complex computing environments
3. Monitoring: the ability to monitor for certain events or conditions
4. Situation Awareness: the ability of a system to provide information about the overall state of the system
5. Scriptability: the ability to script add-ons or automate tasks provided by the system.
6. Logging Information: information that echoes or repeats previous actions taken
7. Accessibility: the ease with which information can be accessed or extracted from the system
8. Accuracy: the user's perception that the information is correct
9. Integration: the way the system allows data and functions to be integrated from various sources
10. Information Completeness: the degree to which the system provides all necessary information
11. Information Format: the user's perception of how well the information is presented
12. Information Currency: the user's perception of the degree to which the information is up to date
13. Speed: the degree to which the system offers timely responses to requests for information or action, including the speed of tool start up/initiation.
14. Reliability: dependability of system operation
15. Verification Information: information that echoes or repeats the outcomes of previous actions taken
16. Trust: the credibility of a system and its output

Upon further inspection, these characteristics seem to fall into categories of attributes pertaining to attributes of the information supplied by the system and attributes of the system itself. This classification of characteristics can be seen in Table 1.

Model and Theory

Although the above list of characteristics important to system administrators is interesting, it does little more than summarize observations and offer untested guidance to practitioners. Without evidence that these characteristics will influence a system administrator to use a particular tool, practitioners will be reluctant to invest the time and money needed to implement these features. The goal of this study is to understand the link between these characteristics and their impact on system administrator perceptions and ultimately, use of the system.

[Wixom and Todd, 2005] present a modification of DeLone and McLean's original user satisfaction model [DeLone and McLean, 1992] that links system and information satisfaction with the behavioral predictors found in technology acceptance literature [Davis, 1989], perceived ease of use and usefulness. They argue that the object-based attitudes and beliefs expressed in system quality, information quality, system satisfaction, and information satisfaction affect the behavioral beliefs that are captured in ease of use and usefulness. These behavioral beliefs, in turn, influence a user's behavior (i.e., their use or non-use of a system). Essentially, this new model represents a theoretical integration of user satisfaction and technology acceptance theories. The strength of the model lies in its ability to guide IT design and development and predict system usage behaviors. System and information quality antecedents offer concrete attributes important to the user that can be addressed and tested throughout the system development lifecycle (see Figure 1).

Because system administrators are still computer users in the general sense, we expect the overall theoretical model to hold. Their unique work environment, technical background and job requirements, however, suggest that they may have different needs when using computers or software applications to do their jobs. Previous studies (e.g., [Bailey and Pearson, 1983; Baroudi and Orlikowski, 1987; Davis, 1989]) have focused on a relatively small number of characteristics that, although telling in their underlying structure [Wixom and Todd, 2005], have been criticized for investigating arbitrary system attributes [Galletta and Lederer, 1989]. The analysis of system administrator work practices above identifies system and information quality attributes (i.e., antecedents) that are meaningful and important to system administrators.

To summarize, research suggests that system administrators may be unique users with system and information requirements that are different from the requirements of regular computer users. We have presented a modified user satisfaction model that links system design attributes to end user satisfaction and system use, presenting an opportunity to measure the impact that these identified attributes have on system administrator beliefs and tool usage. We believe that this model provides researchers guidance for adapting existing user information satisfaction models for tools used by system administrators. Next, we present the methodology used to empirically test the model.

Methodology

System administrators use a self-selected suite of tools to do their work. Our interviews showed that many system administrators within the same organization and even on the same team use different tools and different sets of tools to perform the same tasks. Given this variability of tool choice and use, the difficulty in gathering survey responses from hundreds of system administrators on one particular tool was apparent. As such, we opted to administer the survey to sysadmins of all types (e.g., network administrator, operating system administrator, web administrator, etc.); we asked each participant to identify the tool they used most often in their jobs and complete the survey with that one particular tool in mind. Because the surveys were completed for a tool used most often by the participants, their intention to use the tool is implied; as such, our survey instrument tested all aspects of the model leading up to and including the sysadmin's behavioral attitude towards use of the tool. That is, we did not test the intention to use a tool, because we know the tool is already in use.

Instrument Development

A survey methodology was utilized to collect the data for this study. Once the constructs were identified (i.e., the information and system attributes identified above), corresponding measurement items were researched. When possible, previously validated measures were used. Measurement items for the new constructs (i.e., credibility, scalability, scriptability, situation awareness, and monitoring) were developed following Churchill's [Churchill, 1979] methodology. Items were created based on construct definitions and components identified in the literature. Next, a sorting task was used to determine face and discriminant validity. Each measurement item was written on a 3x5 note card and all cards were shuffled. Three professional system administrators were asked to sort the cards into logical groups and name each group. Each sysadmin sorted the items into the five groups and specified similar identifying terms. Based on participant feedback, the wording on some items was slightly modified. These constructs used a seven-point scale anchored on "Very strongly disagree" and "Very strongly agree," as described above.

Before implementing the survey, paper-based surveys were created with input from colleagues in academics and IT. Next, the instrument was pre-tested with three system administrators. While some wording was edited for clarity, no major issues were reported with the survey instrument. An online version of the survey instrument was then pre-tested by 24 system administrators. Based on feedback and responses to the pilot survey, minor modifications were made. The final survey included 64 items representing the 23 constructs, as well as demographic information. Table 2 summarizes the constructs, number of items, and references.

Sample

To obtain survey participants, an announcement was posted on professional system administrator association message boards (e.g., LOPSA and SAGE) and emailed to participants as requested. In order to reach as many system administrators as possible, participants were also invited to refer fellow system administrators to the study. A web-based survey method was selected

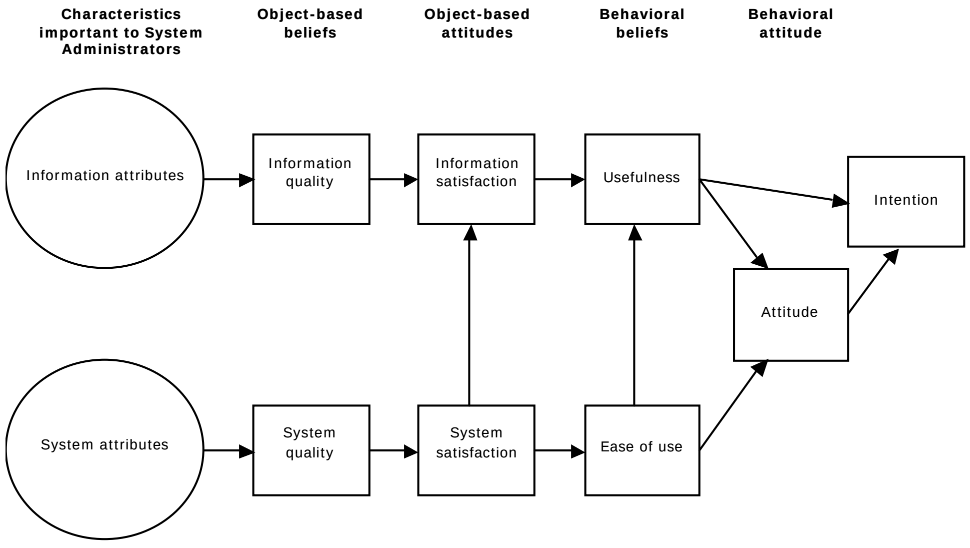


Figure 1: Modified user satisfaction model.

because of ease of distribution and data collection and the targeted respondents' access to the Internet and familiarity with web-based applications and tools.

Survey respondents were professional system administrators who were solicited through professional association message board postings. After removing incomplete responses, 125 surveys were fully completed. The average time to complete the survey was 23 minutes. Of the survey respondents, 91.2% were male and 8.8% were female. The age of respondents ranged from 21 to 62, with an average age of 37.5. Participants reported working at their current organization for an average of 5.40 years (ranging from three weeks to 26 years) and reported working as a system administrator for an average of 12.39 years (ranging from two years to 29 years). Participant demographics were similar to those found in the 2005-2006 SAGE Salary Survey [SAGE, 2006], considered the most comprehensive survey of system administrator personal, work, and salary demographics. These similarities suggest our survey sample is representative of system administrators. Almost half of our survey participants worked for for-profit organizations and companies (49.6%), including manufacturing, high tech, and finance. The next largest number of respondents (38.4%) worked in academic settings, while others worked for non-profit organizations (5.6%), government agencies (5.6%), or in research (0.8%).

Descriptive statistics for the importance of each attribute, as reported by the participants, can be seen below in Table 3.

Results

The strength of the measurement model was tested through its reliability, convergent validity, and discriminant validity. Reliability is established with Cronbach's alpha [Nunnally, 1978] and Composite Reliability [Chin, et al., 2003] scores above 0.70; though Composite Reliability is preferred [Chin, et al., 2003] and Cronbach's alpha can be biased against short scales (i.e., 2-3 item scales) [G. Carmines and A. Zeller, 1979]. Following factor analysis, six items that loaded below the 0.70 level were dropped, resulting in

Constructs	Items	Refs	Constructs	Items	Refs
Completeness	2	W&T	Scalability	3	New
Accuracy	3	W&T	Scriptability	3	New
Format	3	W&T	Situation Awareness	4	New
Currency	2	W&T	Monitoring	3	New
Logging	2	New	Information Quality	2	W&T
Verification	2	New	System Quality	2	W&T
Reliability	3	W&T	Information Satisfaction	2	W&T
Flexibility	3	W&T	System Satisfaction	2	W&T
Integration	2	W&T	Ease of Use	2	W&T
Accessibility	2	W&T	Usefulness	3	W&T
Speed	2	W&T	Attitude	2	W&T
Credibility	5	New			

Table 2: Constructs (W&T = Wixom and Todd, 2005).

Attribute	Minimum	Maximum	Mean	Std. Deviation
Accuracy	3	5	4.74	0.506
Accessibility	2	5	3.98	0.762
Completeness	1	5	3.74	0.870
Credibility	1	5	4.57	0.700
Currency	2	5	4.23	0.709
Flexibility	1	5	3.92	0.947
Format	1	5	3.58	0.900
Integration	1	5	3.50	0.947
Logging	1	5	3.62	0.982
Monitoring	2	5	3.78	0.906
Reliability	3	5	4.68	0.576
Situation Awareness	1	5	3.72	0.876
Scalability	2	5	3.79	0.927
Scriptability	1	5	4.12	0.993
Speed	2	5	3.66	0.782
Usefulness	2	5	4.31	0.745
Verification	1	5	3.38	0.904

Table 3: Importance of attributes identified.

constructs with Composite Reliability scores greater than 0.70, as shown in Table 4. Therefore, our measures are reliable. Convergent validity is established when average extracted variance (AVE) is greater than 0.50 and discriminant validity is established when the square root of AVE is greater than the correlations between the construct and other constructs. Table 5 shows the correlation matrix, with correlations among constructs and the square root of AVE on the diagonal. In all cases, the square root of AVE for each construct is larger than the correlation of that construct with all other constructs in the model. Therefore, we have adequate construct validity.

Discriminant and convergent validity are further supported when individual items load above 0.50 on their associated construct and when the loadings within the construct are greater than the loadings across constructs. Loadings and cross-loadings are available from the first author. All items loaded more highly on their construct than on other constructs and all loaded well above the recommended 0.50 level.

The proposed model was tested with Smart PLS version 2.0 [Ringle, et al., 2005], which is ideal for use with complex predictive models and small sample sizes [Chin, et al., 2003]. R2 values indicate the amount of variance explained by the independent variables and path coefficients indicate the strength and significance of a relationship. Together, R2 values and path coefficients indicate how well the data support the proposed model. User interface type (purely GUI, purely CLI, or a combination of GUI and CLI) was used as a control variable and was linked to both Information Quality and System Quality. A significant relationship was found to System Quality (path = 0.13, $p < 0.05$), but not to Information Quality.

Figure 2 shows the results of the test of the model. All paths in the high-level user satisfaction model are supported. Only four attributes were significant: accuracy, verification, reliability, and credibility.

The results of the test of the research model can be interpreted as follows: Usefulness (0.40) and Ease of Use (0.50) both had a significant influence on Attitude, accounting for 63% of the variance in the measure. Information Satisfaction (0.53) and Ease of Use (0.22) had a significant influence on Usefulness and accounted for 48% of the variance in Usefulness. System Satisfaction (0.66) had a significant influence on Ease of Use and accounted for 44% of the variance in Ease of Use. Information Quality (0.61) and System Satisfaction (0.29) both had significant influences on Information Satisfaction, accounting for 74% of the variance in Information Satisfaction. System Quality (0.81) significantly determined System Satisfaction and accounted for 67% of the variance in that measure. Accuracy (0.58) and Verification (0.22) were significantly related to Information Quality and accounted for 55% of the variance in the measure. Reliability (0.36) and Credibility (0.38) were significantly related to System Quality and accounted for 75% of the variance in System Quality.

	# Items	Cronbach's Alpha	Composite Reliability	AVE
Currency	2	0.77	0.90	0.81
Completeness	2	0.55	0.82	0.69
Accuracy	2	0.63	0.84	0.73
Format	3	0.94	0.96	0.90
Logging	2	0.90	0.95	0.90
Verification	2	0.85	0.93	0.87
Reliability	3	0.90	0.94	0.83
Flexibility	3	0.80	0.88	0.71
Integration	2	0.80	0.91	0.83
Accessibility	2	0.69	0.87	0.76
Speed	2	0.81	0.91	0.84
Scriptability	3	0.86	0.91	0.78
Scalability	3	0.78	0.87	0.70
Credibility	2	0.81	0.91	0.84
Situation Awareness	3	0.78	0.87	0.65
Monitoring	2	0.79	0.88	0.78
Information Quality	2	0.84	0.93	0.86
System Quality	2	0.88	0.94	0.89
Information Satisfaction	2	0.86	0.94	0.88
System Satisfaction	2	0.91	0.96	0.92
Usefulness	3	0.77	0.87	0.69
Ease of Use	2	0.72	0.87	0.78
Attitude	2	0.88	0.94	0.89

Table 4: Reliability and validity analysis.

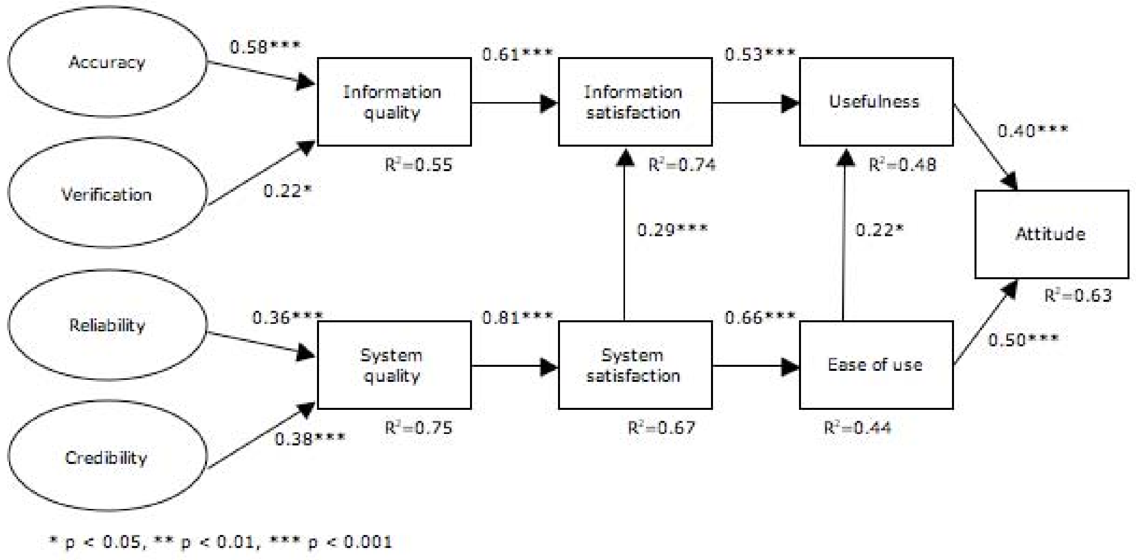


Figure 2: Research model results.

	Accuracy	Accessibility	Attitude	Completeness	Credibility	Currency	Ease of Use	Flexibility	Format	Integration	Info Quality	Info Satisfaction
ACC	**0.85**											
ACCESS	0.51	**0.87**										
ATT	0.55	0.58	**0.94**									
COMPL	0.59	0.64	0.50	**0.83**								
CRED	0.63	0.51	0.66	0.37	**0.92**							
CURR	0.63	0.34	0.25	0.59	0.29	**0.90**						
EOU	0.47	0.59	0.72	0.48	0.54	0.27	**0.88**					
FLEX	0.37	0.42	0.54	0.34	0.56	0.10	0.34	**0.84**				
FMT	0.54	0.58	0.43	0.63	0.29	0.52	0.47	0.05	**0.95**			
INT	0.21	0.46	0.37	0.33	0.27	0.13	0.35	0.54	0.22	**0.91**		
IQUAL	0.70	0.63	0.72	0.52	0.75	0.45	0.59	0.46	0.48	0.38	**0.93**	
ISAT	0.60	0.73	0.73	0.53	0.68	0.37	0.61	0.46	0.49	0.43	0.85	**0.94**
LOG	0.22	0.15	0.24	0.33	0.13	0.21	0.15	0.37	0.25	0.33	0.17	0.12
MON	0.27	0.34	0.23	0.27	0.26	0.29	0.19	0.28	0.15	0.32	0.30	0.27
REL	0.62	0.43	0.63	0.34	0.80	0.27	0.51	0.48	0.27	0.20	0.67	0.59
SA	0.30	0.44	0.32	0.41	0.35	0.32	0.25	0.39	0.20	0.43	0.42	0.46
SCALE	0.43	0.22	0.44	0.27	0.59	0.13	0.33	0.49	0.07	0.19	0.44	0.38
SCRIPT	0.21	0.15	0.36	0.09	0.37	-0.02	0.15	0.77	-0.10	0.49	0.23	0.21
SPEED	0.46	0.38	0.54	0.34	0.54	0.20	0.43	0.43	0.12	0.22	0.52	0.43
SQUAL	0.59	0.46	0.71	0.30	0.80	0.21	0.57	0.60	0.27	0.33	0.74	0.66
SSAT	0.65	0.60	0.85	0.47	0.77	0.31	0.66	0.57	0.41	0.35	0.81	0.78
USEF	0.45	0.58	0.67	0.40	0.63	0.19	0.55	0.60	0.30	0.40	0.60	0.67
VERI	0.15	0.18	0.28	0.27	0.16	0.13	0.17	0.33	0.23	0.33	0.22	0.16

	Logging	Monitoring	Reliability	Situation Awareness	Scalability	Scriptability	Speed	System Quality	System Satisfaction	Usefulness	Verification
LOG	**0.95**										
MON	0.14	**0.88**									
REL	0.18	0.23	**0.91**								
SA	0.16	0.56	0.28	**0.81**							
SCALE	0.11	0.17	0.57	0.22	**0.84**						
SCRIPT	0.46	0.16	0.32	0.22	0.39	**0.88**					
SPEED	0.18	0.21	0.62	0.15	0.42	0.34	**0.92**				
SQUAL	0.24	0.23	0.78	0.25	0.53	0.46	0.57	**0.94**			
SSAT	0.23	0.22	0.75	0.34	0.50	0.37	0.55	0.83	**0.96**		
USEF	0.16	0.31	0.49	0.47	0.42	0.42	0.42	0.57	0.64	**0.83**	
VERI	0.77	0.22	0.18	0.27	0.11	0.41	0.21	0.20	0.21	0.20	**0.93**

Table 5: Correlation between constructs. Bold numbers on the diagonal are sqrt(AVE).

Discussion

These results suggest that at the macro level, system administrators are similar to regular computer users; the user satisfaction model is significant and predictive of their attitude towards computer system use. These results also confirm our intuition that at the micro level, system administrators have specific needs of a computer system that differ from regular users.

When looking at Information Quality, only one attribute found significant in other studies (e.g., [Wixom and Todd, 2005]) was supported, Accuracy. Other attributes previously found significant (Currency, Completeness, and Format) were not. Furthermore, one new attribute was found significant, Verification. Some of these findings may be explained by the work practices of system administrators.

Findings show that accuracy and verification explain 55% of the variance for information quality. Information accuracy is a very real need for system administrators, and was found to be significant in this study. System planning, updating, and debugging is often done with only the information supplied by the system; rarely is a system administrator lucky enough to have a system failure physically apparent, and thus must rely on the accuracy of the information supplied to them. Verification information was found to be a significant influence on information quality. This echoes the findings of the study reported earlier. While a log of previous actions taken on the system may be relatively simple to access, a list of the outcomes of previous actions may be more difficult to generate.

When looking at System Quality, again only one attribute found significant in other studies (e.g., Wixom and Todd, 2005) was supported, Reliability. Other attributes previously found significant (Flexibility, Integration, Accessibility, Speed) were not. One new attribute, Credibility, was found significant.

Findings show that reliability and credibility explain 75% of the variance for system quality. The reliability of a system is of utmost importance; downtime in a large system can cost $500,000 per hour [Patterson, 2002]. It should come as no surprise, then, that the tools used to manage, configure, and monitor those systems need to be just as reliable. The credibility of a tool was also a significant finding in our study. Another study has found similar results [Takayama and Kandogan, 2006], reporting that trust was an underlying factor in system administrator user interface choice.

Conclusions

The purpose of this study was twofold: One, to empirically test the user satisfaction model in the context of system administration, and two, to identify and empirically test system and information attributes important to system administrators. We found that the theoretical model does hold for system administrators, and that they do, in fact, have unique needs in the systems they use.

This study has implications in both tool evaluation and design. By validating the appropriateness of the user satisfaction model in the context of system administration, researchers can utilize this method to evaluate systems. This research has also identified four tool features that are significant to system administrators – accuracy, verification, reliability, and credibility – and should strive to design tools with these attributes in mind.

Author Biographies

Nicole Velasquez is a Post-doctoral Research Associate at the University of Arizona and an enterprise systems tester with IBM. She has experience as a sysadmin, programmer, and systems analyst and earned her Ph.D. in Management Information Systems from the University of Arizona in 2008. Her research focuses on knowledge management systems, information systems success, usability, and system administrators. She can be reached at nicolefv@gmail.com .

Suzie Weisband is an Eller Fellow and Associate Professor of Management Information Systems at the University of Arizona. She received her Ph..D from Carnegie Mellon University in 1989. Her research focuses on collaboration and coordination in face-to-face and computer mediated contexts, with a current focus on the dynamics of large-scale collaborations across multiple people, projects, and resources. She can be reached at weisband@email.arizona.edu .

Alexandra Durcikova is an Assistant Professor of Management Information Systems at the University of Arizona. She has experience as an experimental physics researcher and received her Ph.D. from the University of Pittsburgh in 2004. Her research focuses on knowledge management systems (KMS), the role of organizational climate in the use of KMS, and IS issues in developing countries. She can be reached at alex@eller.arizona.edu .

Bibliography

[Bailey, et al., 2003] Bailey, J., M. Etgen, and K. Freeman, ‘‘Situation Awareness and System Administration,’’ *System Administrators are Users*, CHI, 2003.

[Bailey and Pearson, 1983] Bailey, J. E. and S. W. Pearson, ‘‘Development of a Tool for Measuring and Analyzing User Satisfaction,’’ *Management Science*, Vol. 29, Num. 5, pp. 530-545, 1983.

[Baroudi and Orlikowski, 1987] Baroudi, J. and W. Orlikowski, ‘‘A Short Form Measure of User Information Satisfaction: A Psychometric Evaluation and Notes on Use,’’ http://dspace.nyu.edu, 1987.

[Barrett, et al., 2003] Barrett, R., Y. Chen, and P. Maglio, ‘‘System Administrators Are Users, Too:

Designing Workspaces for Managing Internet-Scale Systems,'' *Conference on Human Factors in Computing Systems*, 2003.

[Barrett, et al., 2004] Barrett, R., et al., ''Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices,'' *Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, pp. 388-395, 2004.

[Bodker, 1989] Bodker, S., ''A Human Activity Approach to User Interfaces,'' *Human-Computer Interaction*, 1989.

[Button and Harper, 1995] Button, G. and R. Harper, ''The Relevance of 'Work-Practice' for Design,'' *Computer Supported Cooperative Work (CSCW)*, 1995.

[G. Carmines and A. Zeller, 1979] Carmines, E. G. and R. A. Zeller, *Reliability and Validity Assessment*, 1979.

[Chin, et al., 2003] Chin, W. W., B. L. Marcolin, and P. R. Newsted, ''A Partial Least Squares Latent Variable Modeling Approach for Measuring Interaction Effects: Results from a Monte Carlo Simulation Study and an Electronic-Mail Emotion/ Adoption Study,'' *Information Systems Research*, Vol. 14, Num. 2, 189-217, 2003.

[Churchill, 1979] Churchill, G., ''A Paradigm for Developing Better Measures of Marketing Constructs,'' *Journal of Marketing Research*, 1979.

[Davis, 1989] Davis, F. D., ''Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology,'' *Management Information Systems Quarterly, Vol. 13, Num. 3, pp. 319-340, 1989*.

[DeLone and McLean, 1992] DeLone, W. H. and E. R. McLean, ''Information Systems Success: The Quest for the Dependent Variable,'' *Information Systems Research*, Vol. 3, Num. 1, pp. 60-95, 1992.

[Fitzpatrick, et al., 1996] Fitzpatrick, G., S. Kaplan, and T. Mansfield, ''Physical Spaces, Virtual Places and Social Worlds: A Study of Work in the Virtual,'' *Proceedings of the 1996 ACM conference on Computer Supported Cooperative Work*, 1996.

[Galletta and Lederer, 1989] Galletta, D. F. and A. L. Lederer, ''Some Cautions on the Measurement of User Information Satisfaction,'' *Decision Sciences*, Vol. 20, Num. 3, pp. 19-439, 1989.

[Haber and Kandogan, 2007] Haber, E. and E. Kandogan, ''Security Administrators in the Wild: Ethnographic Studies of Security Administrators,'' *SIG CHI 2007 Workshop on Security User Studies: Methodologies and Best Practices*, 2007.

[Haber and Bailey, 2007] Haber, E. and J. Bailey, ''Design Guidelines for System Administration Tools Developed through Ethnographic Field Studies,'' *Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology*, 2007.

[HP, 2007] HP, *Adaptive Infrastructure*, 2007, http://h71028.www7.hp.com/enterprise/cache/483409-0-0-0-121.aspx .

[IBM, 2006] IBM, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, 2006, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf .

[Kephart and Chess, 2003] Kephart, J. O. and D. M. Chess, ''The Vision of Autonomic Computing,'' *IEEE Computer*, Vol. 36, Num. 1, pp. 41-51, 2003.

[Nunnally, 1978] Nunnally, J. C., *Psychometric Theory*, 1978.

[Patterson, 2002] Patterson, D., ''A Simple Way to Estimate the Cost of Downtime,'' *Proceedings of LISA '02*, 185-188, 2002.

[Patterson, et al., 2002] Patterson, D., et al., ''Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies,'' Technical Report CSD-02-1175, 2002, http://roc.cs. berkeley.edu/papers/ROC_TR02-1175.pdf .

[Ringle, et al., 2005] Ringle, C. M., S. Wende, and S. Will, ''Smart PLS 2.0 (M3) Beta,'' 2005, http://www.smartpls.de .

[SAGE, 2006] SAGE, *SAGE Annual Salary Survey 2005-2006*, 2006.

[SunMicrosystems, 2006] SunMicrosystems (2006), *N1 Grid System*, http://www.sun.com/software/gridware .

[Takayama and Kandogan, 2006] Takayama, L. and E. Kandogan, ''Trust as an Underlying Factor of System Administrator Interface Choice,'' *Conference on Human Factors in Computing Systems*, 2006.

[Wixom and Todd, 2005 Wixom, B. H. and P. A. Todd, ''A Theoretical Integration of User Satisfaction and Technology Acceptance,'' *Information Systems Research*, 2005.

Dynamic Dependencies and Performance Improvement

Marc Chiarini and Alva Couch – Tufts University

ABSTRACT

The art of performance tuning is, alas, still an art; there are few tools to help predict the effects of changes that are motivated by performance needs. In this work, we present dynamic dependency modeling techniques for predicting the effects of configuration changes. These techniques utilize a simple, exterior model of the system under test, and only require codifying the nature of dependencies between subsystems. With a few examples, we demonstrate how to utilize these models to predict the impact of system changes. Using simple tools and a reasoning process, it is possible to answer many questions about possible performance enhancements. The impact of this work is to advance performance tuning a small amount – from being an art toward becoming a science.

Introduction

This paper was conceived in the context of a very practical problem. In a pilot course on service performance analysis, the class undertook a term project to determine how to improve performance at a reasonably large (3000+ simultaneous users) social networking site. The owner reported that the site exhibits self-limiting behavior, in the sense that the number of simultaneous users is limited by and coupled to server capacity. Thus, our problem was to increase server capacity as much as possible within reasonable economic limits. The site was implemented using a LAMP architecture, with the MySQL and Apache server executing on the same host. The owner of the site proposed an initial alternative, which was to move MySQL to an external enterprise cluster with much more CPU capacity. The initial goal of study was to determine whether this proposal was reasonable and – to the extent possible – to predict the benefits of this change before making it (because such a change is expensive to make).

The students began attacking the problem by understanding traditional performance analysis, as defined in Menascé's books [16, 17]. We found out quickly, however, that traditional methods for performance analysis failed to predict the behaviors we were observing. Part of the reason was that our model of application behavior was incorrect, and there were hidden factors in how the application was written that seriously affected performance. The classical theory thus failed to be useful, and the performance results we actually observed did not look anything like what we might have expected. Artifacts in our observations led us to realize that our model of system behavior was quite näive.

For example, memory caching had a dominant effect upon performance. After studying some confusing measurements, we discovered that the application makes extensive use of in-memory caching of MySQL query results (using the memcache PHP library). The critical bottleneck was not the normally CPU-intensive process of uncached query execution, but rather, memcache's intensive use of memory. We concluded that deploying a clustered MySQL server would have at best a minor effect upon performance: a lot of bucks for little or no bang. A better alternative would be to migrate MySQL onto a single server, move all static content to a third server, and give the dynamic content server as much memory in which to cache MySQL results as possible.

This experience was a hard lesson that changed our thinking about service performance prediction in fundamental ways. In a complex system or application, it can be difficult and costly to develop a detailed model of internals. This greatly reduces the effectiveness of traditional performance analysis theory. Single servers are difficult enough to understand, networks of heterogeneous systems even more. What is needed, therefore, are new theories and methodologies for analyzing performance that do not require detailed internal modeling.

This paper is a small first step toward that goal. We attempt to describe, package, and discuss the methodology by which we came to these conclusions. Nothing in this paper is new by itself; it is the combination of elements into a coherent tuning strategy that is new. We hope that this strategy will inspire other system administrators to both employ and improve it for their needs. The ideas here are a start, but in no sense a mature answer.

Overview

Performance modeling and tuning of a complex system by experimental means has a very different form than classical performance tuning [16, 17]. The approach follows these steps, which we discuss in further detail in the remaining sections:

1. *Factor* the system under test into independent subsystems and resources subject to change.
2. *Synthesize* steady-state behavior via an observation plan.
3. *Determine* dependencies between performance and resources.
4. *Validate* the dependency model by observation.
5. *Reduce* resource availability temporarily.
6. *Measure* the difference in performance between current and slightly-reduced resources.
7. *Compute* critical resources from the measurements.
8. *Expand* the amount of a critical resource.
9. *Repeat* the methodology for the expanded system to note improvement (or lack thereof).

This is nothing more than a structured scientific method. Some of these steps are intuitive, and some require detailed explanation. The main difference between this and trial by error is that we are better informed, and the path to our goal is more strictly guided.

The reasons we present this methodology are threefold. First, although there is a trend toward open-source in IT, there remain a large number of inscrutable closed-source software systems. Detailed modeling (and understanding) of the interaction between such systems is impractical for system administrators. Second, SAs should not need to be software engineers or computer scientists in order to squeeze optimal performance out of their infrastructure. They require tested, transparent, and relatively intuitive techniques for achieving performance gains without greatly sacrificing other goals. Last, the rapid adoption of virtualization will significantly increase the complexity and opacity of performance problems. External modeling will become a necessity rather than an alternative.

In the following, we mix intuition and previously developed rigorous mathematics to inform our approach. It turns out that most of an expert practitioner's intuition about performance tuning has its basis in rigorous mathematics. The flip side of this, though, is that other results arise from the same mathematical basis that are not intuitive to system administrators. So, many of our claims will seem like common sense at the outset, but by the end of this paper, rather useful and counter-intuitive claims will arise.

Troubleshooting Performance

A common question asked by users and IT staff alike is "exactly what is making this service run so slowly?" Classical performance analysis [16, 17] relies on the ability to describe a system under study precisely and in detail. In many environments, this can take more time than it is worth, and obtaining sufficient precision in the description may well be wasteful[1] or impossible (because of technical limitations, closed source, or intellectual property concerns).

If we instead look at the system as a collection of interconnected components, whose internal function remains unknown but whose interdependence is understood, we can make inferences about performance without referring to internals. Since the true service model of some components may well be unknowable, this approach sidesteps knowledge of source code and specific service configurations. Thus, it is equally applicable to open source systems as to, e.g., proprietary SANs or turnkey solutions.

Factoring a Service Into Components

The first step in our methodology is to create a high-level model of a service's dependencies about which we can reason. There are two useful ways to think about relationships between a set of interdependent components: as a set of information flows and as a set of dynamic dependencies. The flow model is more familiar; interdependence arises from request/response behavior. Sometimes, however, requests and responses cannot be easily described, and one must instead assert a dependency without an explicit (and measurable) request/response model.

Modeling Information Flows

Traditional performance analysis describes systems via request flow models. For example, how does a web request get served? First, it arrives at the web server, which makes a request for a file, which is serviced by a file server or filesystem, which returns the file, which is processed in whatever manner the web server wishes, and the results are returned to the user. Thus the model of information flow looks like that in Figure 1.

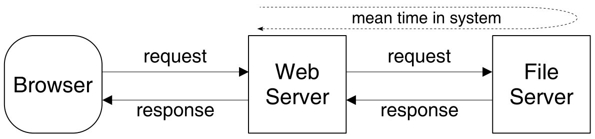


Figure 1: A model of information flow between a web server and its associated file server.

A few basic concepts of this flow model will prove useful. The *mean time in system* for a request is the average amount of time between when a request arrives and the response is sent. What we normally refer to as *response time* is mean time in system, plus network overhead in sending both request and response. In the above diagram, there are two environments in which mean time in system makes sense: the web server and the file server.

It is often useful to express times as their reciprocals, which are rates. The reciprocal of mean time in

[1]We might even say that detailed classical analysis of a performance problem is something like "fiddling while Rome burns." When there is a performance problem, the city is burning down around us and we need to take action. Redrawing a city plan is usually not the first step!

system (in seconds) is *service rate* (requests serviced per second); this is often notated as the symbol μ. The *arrival rate* λ for a component is the rate at which requests arrive to be serviced. Both μ and λ can change over time.

It is common sense that for a system to be in a *steady state*, the arrival rate must be less than the service rate; otherwise requests arrive faster than they can be processed, and delays increase without bound. The concept of steady state is often notated as $\lambda/\mu < 1$.

Steady-State Behavior

The average behavior of a system has little meaning unless the system is in some kind of steady state. The actual meaning of *steady state* is somewhat subtle. We are interested in response time for various requests. Steady state does not mean that response times are identical for each request, but that their statistical distribution does not vary over time. A system is in steady state between time X and time Y if the population of possible response times does not change during that period.

To understand this concept, think of the system being tested as analogous to an urn of marbles. Each marble is labeled with a possible response time. When one makes a request of the system, one selects a marble and interprets its label as a response time for the request. After picking many marbles, the result is a statistical distribution of response times (e.g., Figure 2). More common response times correspond to lots of marbles with the exact same label.

The exact definition of steady state for our systems is that the distribution of possible response times does not change, i.e., there is only one urn of marbles from which one selects response times. No matter when one samples response time, one is getting marbles from the same urn which in turn means that the frequencies of each response time do not vary.

Synthesizing Steady-State Behavior

A real system is seldom in a steady-state condition. Luckily there are ways of obtaining steady-state data for a system in flux by normalizing performance data.

Suppose, for example, that the load on a system changes with the time of day. This is often referred to as a *sinusoidal arrival rate*. A very common technique for dealing with time-varying behavior is to note that sinusoidal arrival rates (e.g., correlated with business hours) can be viewed as steady state if one looks at them for long enough in time. An hour's results may always vary, but when looking at several days worth of data, starting at the same time each day for beginning and end, frequency diagrams look the same regardless of the day upon which one starts.

Likewise, we can also normalize our data into a steady state by sampling behavior that just happens to match in state. Looking at the same hour of the day for several days yields steady-state data. We used this method previously in [8] to model costs of trouble ticket response, by treating each hour of the day as a different sample and combining data for different days (excluding holidays).

Measuring Response Time

There are several ways in which to measure service time. Ideally, it is measured from the point at which a request arrives to the point at which a response is put on the wire. Calculating true service times may or may not be technically feasible, depending on the capabilities of the operating system and/or

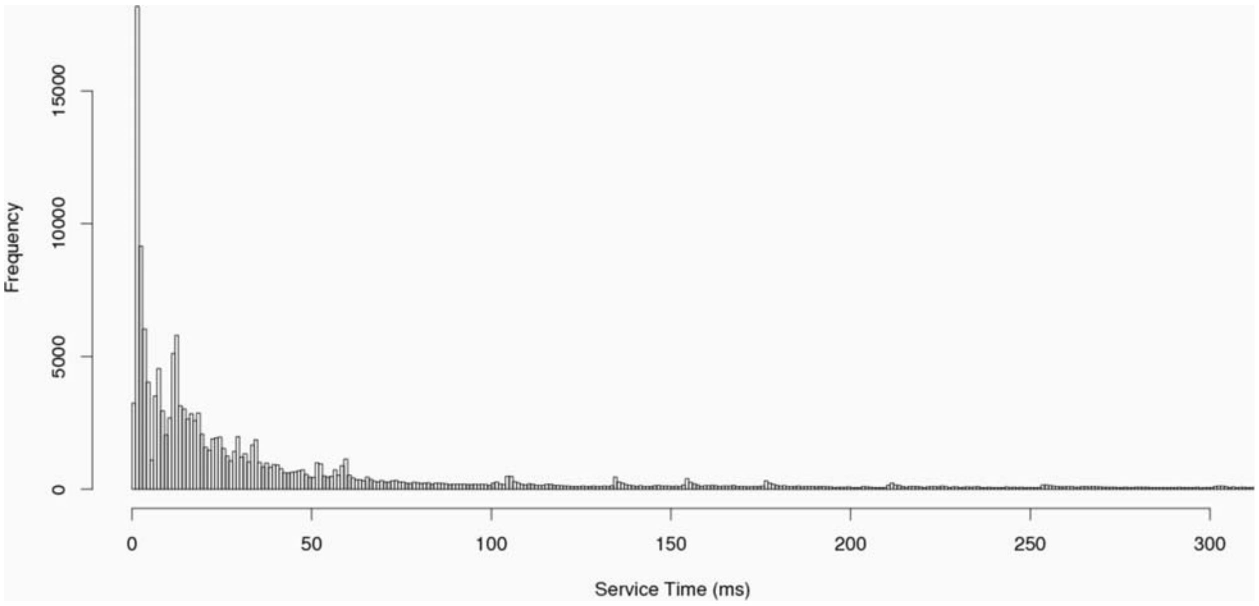


Figure 2: Histogram of response times for multi-class requests over a six hour period. The X axis corresponds to response time while the Y axis represents frequency of that time.

service software. In this work, technical limitations of Apache prevent us from using this measure. Even measurements of socket durations from within the kernel are asymmetric in the sense that while the start time for service is very close to correct, the end time includes the time the response spends in transit. In practice, we can approximate the true service time by sending requests over a high-speed LAN and measuring round-trip times from the source.

To measure response times, we use a benchmarking tool called JMeter [2]. The tool allows us to programatically construct various numbers and types of HTTP requests for static or dynamic content. These requests can then be generated from multiple threads on one or more clients and sent to one or more servers. Additionally, JMeter affords us very fine control over when and for how long tests and sub-tests are run, and the kind of results they collect.

So far, we have run a multitude of simple tests that repeatedly ask a single Apache server for one or more of several *classes* (i.e., file size) of static content. Thus, the components of the service all reside on the same machine. There is no reason, however, that the black box cannot be expanded to include multi-system services or entire networks.

One very useful way to visualize performance of a complex system is via histograms, with response time on the X axis and frequency of that response time on the Y axis. Figure 2 shows a histogram for one 6-hour test. Here, the JMeter client repeatedly asks for a random file from a set of 40 classes. Requests are sent from the client at an average rate of 16.6 per second (Poisson delay of 60 ms). The distribution of service times seems to be exponential, but it is dangerous to conclude this just from eyeballing the graph, especially in the presence of the self-similar ''spikes.''

We cannot say much about the server just by collecting response time statistics. The server's performance depends in large part on which resources it is being asked to utilize. It is impossible to get a picture of every possible combination of requests. To approach the problem in this manner would limit us to making broad statements about our system only as it is currently configured and under arbitrarily determined categories of load.

We can gain more useful insight by normalizing our performance data. Suppose that we have a very simple probe (akin to a heartbeat monitor, but yielding more information) that continually sends requests to the server at a rate that is intended not to affect performance. At regular intervals, we create histograms of the probe results and observe how they change, e.g., calculate their means, medians, standard deviations, etc. If the system is in a steady state, we can expect that the response time distribution will not change much. As saturation approaches, certain probe statistics will change.

Figure 3 shows response time distributions from probes under increasing background loads. In all cases, not only do statistics such as the mean change, but the probability mass changes shape also. This indicates that the underlying distribution is changing, making it difficult to compare against any theoretically ideal baseline. If we determine that the ideal

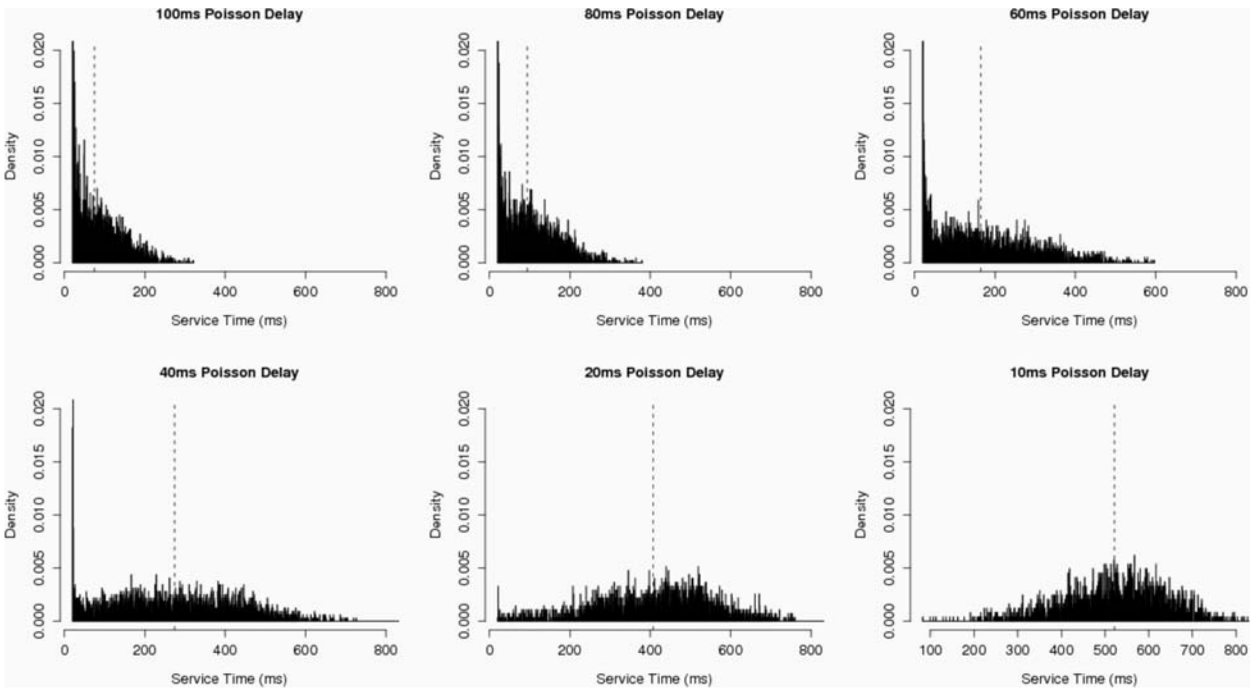


Figure 3: Decreasing the delay between requests (increasing load) has a noticeable effect on the probe's response time distribution. The mean is indicated by the dashed line.

distribution is exponential, what does an observed bimodal or normal distribution tell us about resource utilization? Any goodness-of-fit test for the exponential – which we might use to measure the distance from expected – will fail miserably. This fact motivates our use of non-parametric statistical methods (presented in a later section), which make no assumptions about underlying distributions.

Modeling Dependencies

The nature of information flow within a system determines one level of performance dependencies between subsystems. Our next step is to codify dependencies based upon contention for resources. These dependencies may be more difficult to directly observe; individual requests and responses may be hidden from the analyst. Examples of contention dependencies include the use of memory, CPU cycles, or a network interface among several processes. For these dependencies, there is no (reasonably) measurable notion of a request or response; the system simply slows down (sometimes mysteriously) when there is not enough resource to go around. The reason why something like memory contention is not measurable in terms of a request-response model, is that the act of measurement would slow the system down too much to be practical. The requests and responses do exist (as calls to, e.g., sbrk), but we must theorize their presence without being able to observe them directly.

One way we can begin to model these dependencies is via a *dependency graph*. A dependency graph for a system consists of nodes and edges, where a node represents a resource and a directed edge represents a hypothetical dependency between two resources. $A \to B$ means that A depends upon something that B has.

For example, suppose that the web server and the file server both run on the same host. Then the information flow is just one kind of dependency, and there are three other bounded resources (CPU, disk I/O, and memory) that the servers share. To depict this, we can draw arrows from services to dependencies, as in Figure 4. In this case $A \to B$ means that A depends upon B in some way. Two arrows to the same place indicate possible contention. One value of such a model is that it can describe, at a very high level, the potential performance bottlenecks of a system, in a manner that is invariant of how the system is actually implemented.

The main purpose of a dependency model is to list – in a compact form – the resource limitations that can affect performance. A useful model describes elements that are subject to change, and ignores elements that one cannot change. For example, if one can expand the memory in a system, but cannot change the processor, modeling in-processor cache size is not particularly useful. In practice, this tends to keep dependency models to a manageable size.

The dependency model corresponds to a flow model, of course, but the details of that underlying flow model may not be useful. For example, disk I/O indeed consists of requests and responses but the overall response time of the system is only vaguely related to disk response; there are many other factors that influence performance. Likewise, use of memory corresponds to requests for more memory and responses that grant access.

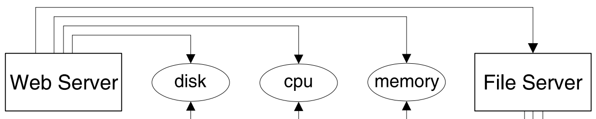


Figure 4: A model of resource conflicts, where two arrows to the same place represent a conflict.

Näive Physics

Dependency models mostly help us reason about performance. The most immediate and easy thing one can do with a dependency model is to utilize what might be most aptly called *näive physics* [20]. The term first arose in the context of artificial intelligence, as a machine model of human intuition in understanding physical processes. A so-called näive model concerns order but not quantity. We know, e.g., given the above model, that if we decrease the memory size, both web server and file server are potentially adversely affected. If we increase memory size, they both potentially benefit. But more important, we also know that if we increase memory size and performance does not improve, then there is some other factor that affects performance that we have not changed. Worse yet, if we change nothing at all and performance worsens or improves, then our model itself is incomplete and/or invalid.

Näive physics are our first defense against formulating an invalid model of a system. When load increases, service time should increase accordingly, and certainly should not decrease. A good understanding of what a decrease or increase should do is enough to allow us to check many simple dependency models.

Validity

Obviously, a dependency graph that omits some important element is not particularly useful. It is best to consider a dependency graph as a hypothesis, rather than a fact. Hypotheses are subject to validation. In this case, the hypothesis can be considered valid (or complete) if changes in resources available within the graph cause reasonable changes in performance at the edges of the system, and invalid (or incomplete) otherwise.

For example, consider a web server. We might start with the initial hypothesis that the web server's performance depends only upon a file server, as above. This means that as file server load goes up for whatever reason, we would expect from näive physics that the web server's performance would decrease or –

at the least – remain constant. Likewise, we would expect that if the file server remains relatively idle, web service should not vary in performance.

Testing this hypothesis exposes a dichotomy between validating and invalidating a hypothesis that will affect all rigorous thinking about our problem. Our hypothesis tests true as long as web server performance *only decreases* when the file server is loaded, and tests false if there is any situation in which, given the same input request rate as before, the file server is not busy and the web server slows down anyway. A point to bear in mind is that hypothesis testing can only ever invalidate a hypothesis; good data simply shows us that the hypothesis is reasonable so far, but cannot prove the hypothesis to be valid.

One potential problem in our model above is that web servers do not necessarily depend just upon file servers. Changing a single line of code in its configuration makes an Apache web server depend additionally (via flow) upon a DNS server to look up all request addresses on the fly. In the context of our model, such a web server must have a *hidden dependency* upon the DNS server. If the DNS server is now explicitly included in our model, then a slowdown in that component offers alternative evidence for why the web server is slow.

Criticality

With a reasonable model in hand, an obvious question is "which elements of a dependency graph should be improved to make the service respond faster?"

A *critical resource* is one that – if increased or decreased – will cause changes in overall performance. For example, available CPU time is usually critical to response time. If there is more than sufficient memory, then changing memory size will not be critical to performance. If our web server is executing many scripts, CPU time may be the most critical resource, while if it is performing a lot of disk I/O, disk read and write speed may be more critical.

Micro Resource Saturation Tests

Testing criticality of a resource in a complex system is difficult. We borrow an idea from [5], used previously to trace packets statistically in a complex network. A resource is critical to the extent that changing its availability or speed changes overall performance. If we can perturb the system enough to change the availability of one resource, and the whole system's performance responds, then the resource is critical. The extent to which a resource is critical is – in turn – determined by how much of an effect a small change has upon overall performance.

We can analyze the effects of resource perturbation via use of a Micro Resource Saturation Test (mRST). A mRST is a programmatic action that – for a little while – decreases the amount of resources available. We cannot change the amount of memory a system has on the fly,[2] but we can – for a short time – consume available memory with another process. We cannot change network bandwidth on the fly but we can – for a short time – disrupt it a bit by doing something else with the network. If response time consistently changes for the service itself in response to our disruptions, then we can conclude that we are changing the amount of a critical resource, and the extent of the change indicates how critical. Using a simple probe, we can sometimes determine which resources should be increased and which are of no importance.

The reader might ask at this point why we do not simply check the available amount of the resources in which we are interested. The reason is that the kernel's idea of availability may well be biased by what one or more processes are doing. For example, in our first case study above, the web server immediately allocated all available memory, so that the memory always looked full. The reality was that this memory was a cache that was only full for part of the time.

Quantifying Criticality

The purpose of a mRST is to perturb a resource r by an amount Δr and observe the difference in performance Δp. Thus the criticality of a resource r under specific load conditions is a derivative of the form:

$$C_r = \frac{dp}{dr} = \lim_{\Delta r \to 0} \frac{\Delta p}{\Delta r}$$

where p represents service performance, Δp represents the change in performance, r represents the amount available of a resource, and Δr represents the change in a resource. This can be estimated as

$$C_r \approx \frac{\Delta p}{\Delta r}$$

for a specific Δr we arrange, and a specific Δp that we measure.

Note that for a multiple-resource system with a set of resources r_i, criticality of the various resources can be expressed as a gradient

$$\nabla p = \Sigma_i \frac{\partial p}{\partial r_i} \approx \Sigma_i \frac{\Delta p}{\Delta r_i}$$

This suggests that one way to tune systems is via gradient ascent, in which one increases a resource in the direction of maximum performance improvement.

Resource criticality is a concept that is only reasonably measured relative to some steady state of a system (in which input and output are in balance). Obviously, resources are only critical when there is work to be done, and the above gradient only makes sense when that work to be done is a constant.

Cost Versus Value

Remember that the objective here is to get the best performance for the least cost. Thus criticality is best defined in terms of the cost of a resource, rather than its value. One can think of the units for r in $\partial p/\partial r$

[2]The advent of virtualization does make this possible, but it is not likely to be practical in a production environment.

as a cost increment rather than a size increment, so that the units of the derivative are *rate/cost* [8]. Often, this is a finite difference because cost is a discrete function, e.g., memory can only be expanded in 128 MB chunks.

Interpreting mRST Results

Critical resource tests can be interpreted either via näive physics or statistics. From a näive physics point of view, resources are either non-critical ($\partial p/\partial r \approx 0$) or critical ($\partial p/\partial r > 0$), where p is the average response rate (1 divided by average response time). Increases in p represent increased performance and increases in r represent increased capacity.

The näive view may suffice for most practice, but we would be served better by a more rigorous approach. From a statistical point of view, some resources can be characterized as more critical than others, but the definition of criticality is rather tricky

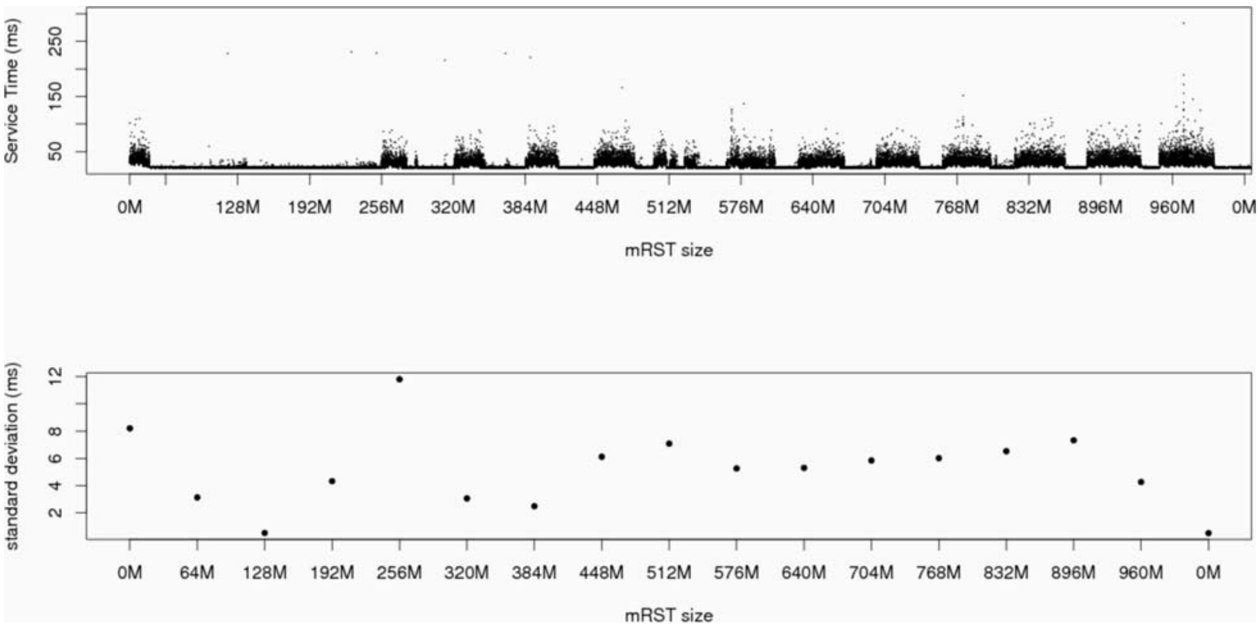


Figure 5: (top) The raw response times of an Apache web server during and between increasing memory mRSTs. The initial perturbation at 0M is due to the loading of the operating system's page-cache. (bot.) The variability (standard deviation) of response times does not change much from test to test.

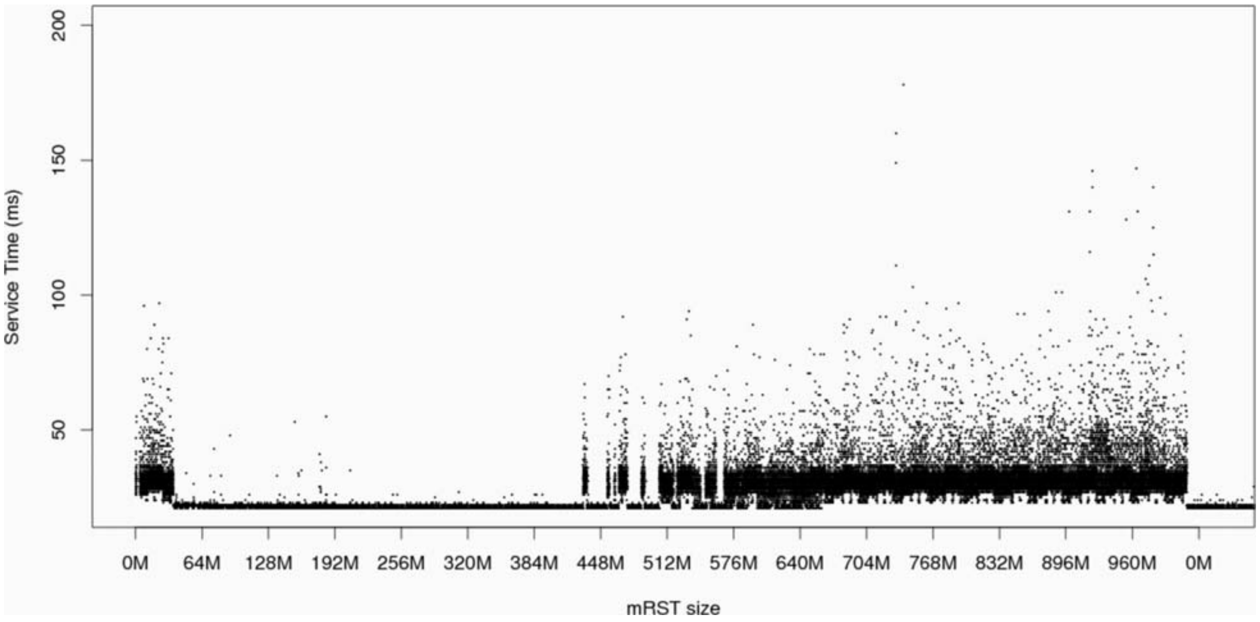


Figure 6: Adding to memory usage with a series of increasing mRST loads shows a clear plateau where memory becomes critical.

and the actual inferences somewhat more subtle than when utilizing näive physics.

Example: Perturbing Memory Use

We can see the effects of perturbation through a simple example. We synthesize steady-state behavior by creating a constant load of probe requests on a server. Figure 5 shows the raw response times of the server when hit with a sequence of memory perturbations. Our test run continually requests a sequence of different files totaling 512 MB in size. The server is equipped with 1 GB of RAM and no swap space. The initial variations in response time (at 0 MB) are a result of the OS filling the page-cache from disk. Each mRST works by obtaining a specified amount of memory and freeing that memory after ten minutes. The lulls (reductions) in response time between perturbations are an indication that the server has returned to an unstressed state and is serving the requested files from cache. There is a relative lack of variation in response times after the page-cache has loaded and up until the 256 MB mRST. This indicates a critical point in memory usage. Also, while there is no specific trend in the standard deviation of the response time (aside from an anomaly at 320 MB), the settling time back to undisturbed performance increases with the size of the mRST. This can be observed in the increasing width of the perturbations. Alternatively, one can observe the same critical point for memory without waiting for the system to reset (Figure 6).

The other case – in which the resource under test is not critical – is shown in Figure 7. In this case, 10 minutes into the test, 512 processes are spawned to compete for CPU cycles for the next 10 minutes. This does not affect response time noticeably, so we conclude that CPU cycles are not critical. Note that this test is much more convincing than load average, because the interpretation of load average changes depending on the OS platform and the architecture of a given machine. In this case, the load average becomes very large, but the system still responds adequately. One must take care, however, when systems are near saturation. Figure 8 shows a series of increasing mRSTs after a system is already saturated. The perturbations in this case do not have a significant effect upon behavior, as shown by the histograms found in Figure 9, which are virtually identical.

Reasoning About Criticality

Reasoning about what is critical is sometimes difficult and counter-intuitive. The system under test is in a dynamic state. It is important to distinguish between what one can learn about this state, and what aspects of performance are unknowable because there is no mechanism for observing them, even indirectly.

For example, the question often arises as to what to change first in a complex system, in order to achieve the most improvement. Testing via mRST does not tell us how much improvement is possible, but does tell us the rate of change of improvement around current resource bounds. If there is no change, a resource is not critical and need not be enlarged and/or improved. After an adjustment, another set of mRSTs is required to check criticality of the new configuration.

For example, suppose that we manage to take up 10% of 1 GB of memory for a short time and overall performance of the system under test decreases by 5%. This does not indicate exactly what will happen if we expand the memory to 2 GB, but instead, bounds any potential improvement in performance. If memory is the only factor, we could perhaps expect a performance

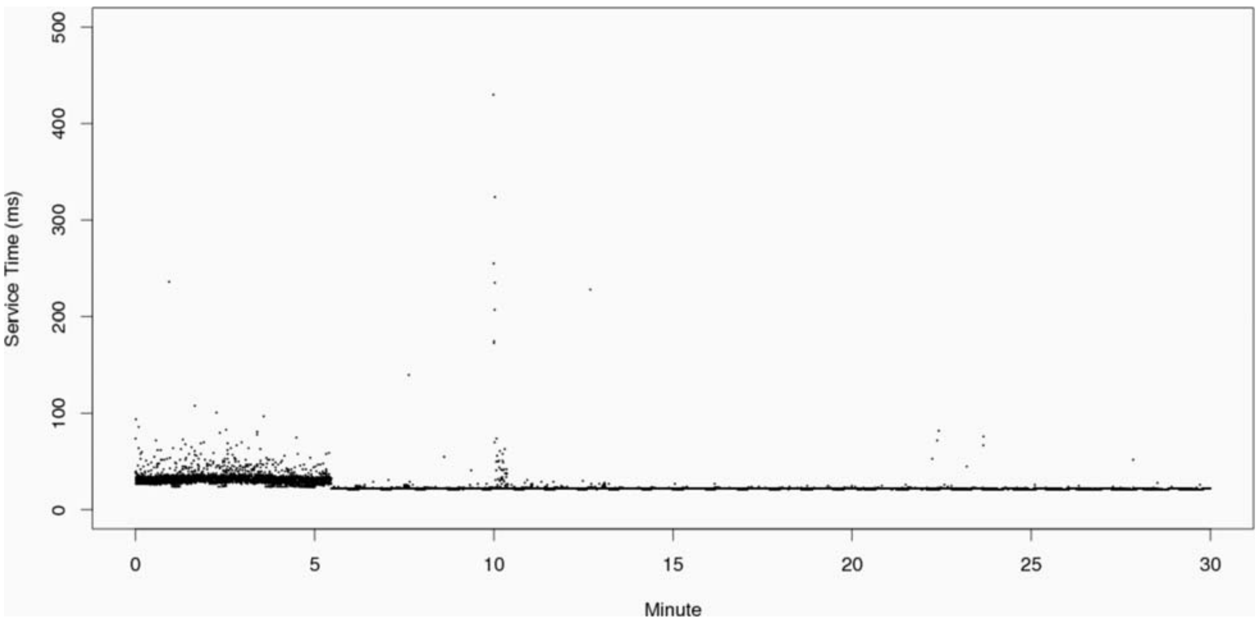


Figure 7: A CPU-time mRST conducted 10 minutes into a test shows almost no criticality of CPU resources.

improvement of 50% (presuming 5% per 100 MB) but this is unlikely. It is more likely that memory will cease being a constraint and another resource (e.g., CPU) will become most critical long before that happens, and that the overall benefit from memory increases will plateau.

To better understand this concept, note that the measurements we can make via mRST are only *point rates* and not global rates. We can measure what will happen to performance if we make a small change in a resource, but this estimation only applies to changes near the estimation. In other words, if we manage to temporarily consume 1 MB of 1024 MB of memory, and performance drops by, e.g., 5%, we can only conclude that adding 1 MB to an existing 1023 MB might improve performance by 5%, and not that adding 1 MB to 1024 MB might improve it by another 5%. We

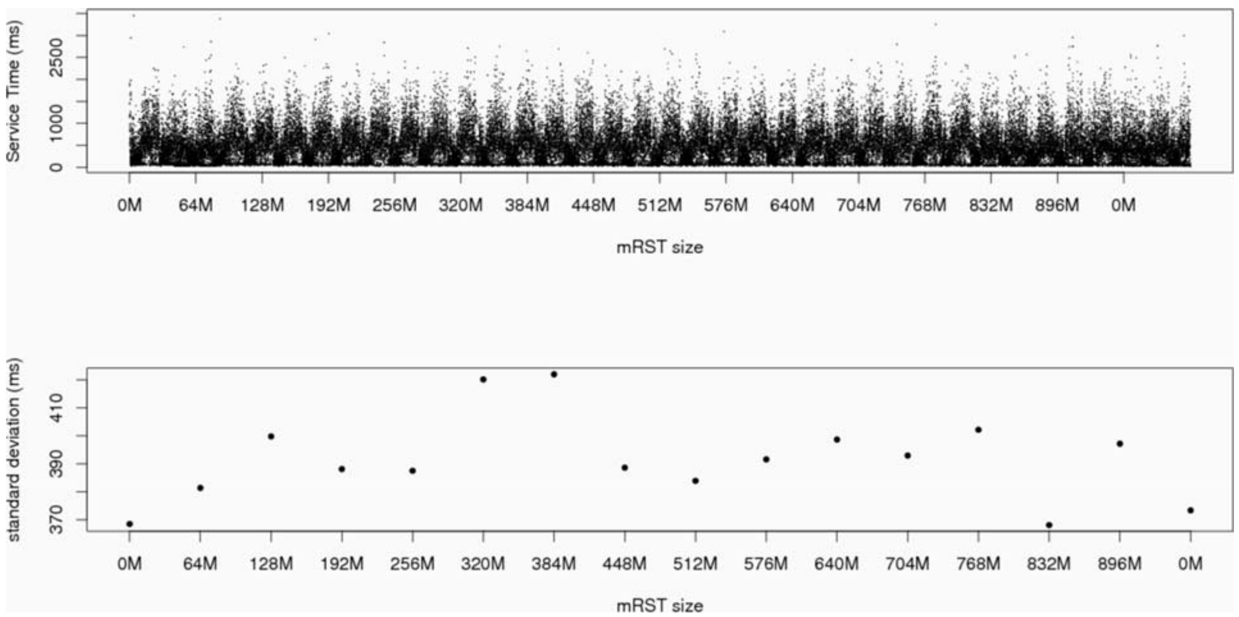


Figure 8: (top) Increasing sequence of mRSTs after saturation with no lulls between increases shows no significant change in behavior. (bot.) Variability in response times during each mRST is more significant than in the case with resting time.

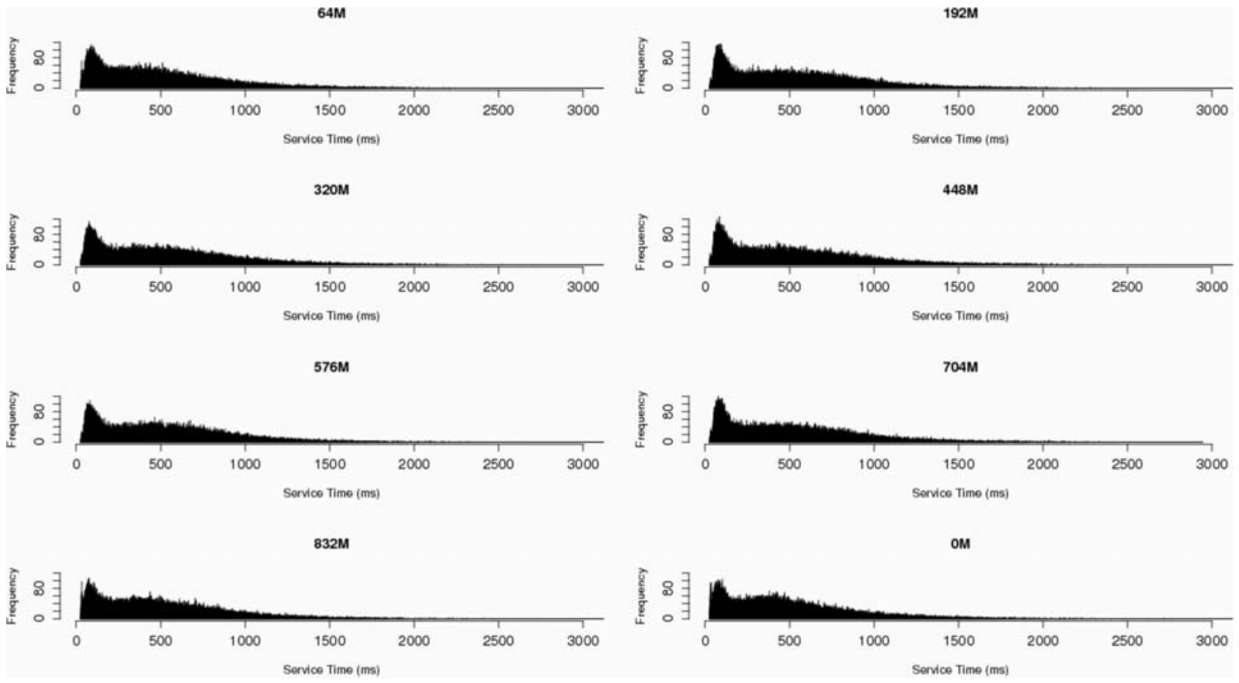


Figure 9: Histograms of response time for progressively larger memory mRST of an already saturated system shows that there is little change in distributions as resource availability decreases.

can be reasonably sure that adding 1 MB to 1024 MB will improve things a little, but we have no information (from the test in hand) as to when we will have added enough memory that the addition of more will no longer be critical.

Statistical Inference

So far, we have discussed a methodology based upon a mix of direct measurement and intuitive reasoning. While this is certainly better than trial by error, there are also powerful mathematical tools we can bring to bear to make this process more rigorous and accurate.

The long-term goal of this work is to develop a cogent statistical inference methodology that allows one to infer – with reasonable certainty – whether system attributes are critical or not. We would like, e.g., to be able to say that "with 95% certainty, resource X is critical." An ideal technique would also measure the extent to which X is critical.

The key question when we observe a change in a system is whether there is a real change. Suppose that we observe an increase in mean response time between two experimental conditions. It could be that this is a true underlying change, but it is also possible that the difference in measurements is due to random factors other than resource criticality.

Returning to the urn analogy, if we measure response time under two conditions, then the basic question is whether there is one urn or two. If the conditions are different, then this is analogous to two different urns from which marbles are being selected. If they are the same, this is analogous to drawing both sets of marbles from the same urn. We need to be able to determine, to some acceptable degree of certainty, whether there are really two different urns.

The key to answering this question is to consider chance. At any time, any marble in the urn could be selected. Thus it is possible, though not very likely, that one could select the same marble over and over again, ignoring the others. This would lead to a measured response time distribution with a different mean, even though we are choosing from the same urn as before.

There are many tests that statisticians use to compensate for randomness in data. Each test utilizes some model of how randomness can arise in the data, and allows one to compute the probability that two sample distributions differ by chance even though the distributions are collected under the same conditions. This probability allows one to judge whether an observed difference is likely to be large enough to matter.

Our tools for this task are non-parametric statistical models and tests [21] that do not assume any particular distribution of data. As illustrated in the figures above, the data we observe has no clear relationship to well-known statistical distributions. Moreover, the true population distribution likely changes under different circumstances. This is important because if we were to use traditional parametric tests such as, e.g., Pearson's chi-square, on data that are not normally distributed, the results would be unreliable or inconsistent at best.[3] The goal of the following section is to compute the probability that there is one urn rather than two. If this probability is high, one cannot confidently claim that observed differences are meaningful; in statistical terms, the differences would not be *significant*. Fortunately for us, this problem arises frequently in the social sciences and we can borrow tools from statistical analysis to help. In order to do this, however, we must carefully analyze our experiments and data, and select tools that apply to the task.

Non-parametric Statistics

One key to applying a statistical technique is to first consider the assumptions required and ensure that it is safe to make those assumptions about one's data.

Statistical techniques that show promise for determining criticality of a resource include the Mann-Whitney-Wilcoxin rank-sum test (U test) [14], and the Kolmogorov-Smirnov test (K-S test) [7]. These tests are robust in the sense that they still perform well when certain distributional assumptions are violated. They are also generally less powerful than their parametric counterparts, which means that they are more likely to generate false negatives [19]. There are ways of improving this, but that discussion is beyond the scope of this paper.

Both of these tests take as input two sample distributions of data, and test whether differences between the samples could have arisen by chance. The output of the test is a *p-value* that describes how likely it is that observed differences are due to chance alone. A low p-value (e.g., $p \leq 0.05$) is evidence that the populations from which the samples are drawn differ, but a high p-value does not give evidence that the distributions are the same. This counter-intuitive result partly arises from the fact that we are sampling the populations; the fact that a particular sample of a population conforms does not mean that the entire population conforms, but if two samples differ sufficiently, this is evidence that the populations differ as well.

To apply these tests meaningfully, however, one must satisfy their requirements. Both the U test and K-S test require that individual samples (e.g., the response times for single requests) are independent. Two samples are independent if the response time for one cannot affect the other's response time, and vice versa. Unfortunately for us, independence does not arise naturally and must be synthesized. Two requests that occur close together in time are not independent, in the sense that they may compete for scarce resources

[3]If the population can be made to approximate a normal distribution by, e.g., collecting large enough samples, then the test would work correctly. However, to make our methodology as efficient as possible, one of our goals is to minimize sample sizes while still obtaining valid results.

and affect each other. But, if one considers two requests that are far enough apart in time, say, 10 seconds, there is no way in which these can compete, so these can be considered independent.

We can use the non-parametric tests in our performance analyses as follows. First we collect data for a steady state of a system under test. We then perturb the system by denying access to some resource and collect data during the denial. These two data sets are both discrete distributions of service response times. We apply a test to these data. If the p-value for the test is $p \leq 0.05$, we can conclude that the two distributions are different, while if $p > 0.05$, results are inconclusive and the difference could be due to randomness alone.[4] Here great care must be taken to avoid erroneous conclusions. Suppose we run the K-S test on two sets of measurements known to be from the same population and they test as significantly different. We know that the K-S test requires that the system be in a steady state, and that the measurements are independent of one another. Thus if the test returns a result we know to be false, then something must be wrong with the way we are applying it: either the system is not in a steady state, or measurements are not independent.

We can exploit this reasoning to test independence of measurements. If we apply the K-S test to two samples from one population, and we know the system is in a steady state, and the test finds significant differences, then we can conclude that either measurements are not independent or the sample size is too small, because all other assumptions of the test are satisfied and the test should not identify significant differences.

Thus we adopt a methodology of grounding our assumptions by synthesizing an environment in which our tests do not flag samples of the baseline steady state as significantly different. To do this, we:

a) probe the system regularly at intervals long enough to assure independence of measurements.
b) validate our probes by assuring that two sets of measurements for the same system state could occur by random chance.
c) create another system state via mRST.
d) check whether that state yields a significantly different population of probe measurements.

The point of this technique is that a system is defined as having a steady state if any deviations in measurements could occur by random chance, so that comparisons between that state and any other then make sense. Otherwise, we cannot be entirely sure that we are applying the test correctly.

Related Work

Systems performance analysis has been conducted for many years and in many contexts, such as hardware design, operating systems, storage, and networking. Many books have been written that focus on the practical sides of analysis such as experimental design, measurement, and simulation [10, 13, 16, 17]. These present comprehensive, general-purpose methods for goals such as capacity planning, benchmarking, and performance remediation. The scope of our research is more narrow; it is less about traditional performance analysis, and more about discovering performance dependencies. It is targeted to a system administration community that is frequently asked to perform small miracles with a deadline of yesterday and at the lowest possible cost.

Several past works have attempted to describe what is normal "behavior" or performance for a system, either for specific classes of load or in a more general fashion [4, 6, 12]. Detecting behavioral abnormalities is also frequently studied in a security context [9] and for fault diagnosis [11, 15]. Since our own work relies on perceived or statistical changes in system response, it is to a significant extent, compatible with many such approaches. In fact, time series analysis techniques such as those found in [4] could be used inside our methodology, but we can make no claims about their efficiency in discovering critical resources.

Finally, prior work that uses black-box testing to uncover performance problems [1, 3, 18] is similar to our own in philosophy, but not in purpose. These research projects attempt to locate nodes that choke performance in a distributed system by analyzing causal path patterns.[5] Our methodology can be adapted to function on larger black boxes such as a distributed system simply by scaling, i.e., we can factor a distributed system into distinct communicating subsystems, e.g., nodes or groups of nodes, and/or resources, e.g., individual services. Once we have located problem nodes, we can apply the process again to the resources on those particular nodes.

Future Work

This paper is just a start on a rather new idea. We plan on eventually embodying the thinking presented here into a toolkit that can aid others in doing performance analysis of this kind. This requires, however, much more groundwork in the mathematics of statistical inference, so that the toolset can enable appropriate statistical reasoning with little or no error or misinterpretation. Hand-in-hand development of tests and statistical reasoning will be paramount.

Conclusions

Resource dependency analysis and performance tuning of complex systems remains an art. In this paper, we have made a small start at turning it into a science. Our initial inspiration was that classical methods of performance analysis that require a complete factoring of a system into understandable and modeled

[4]A 0.05 significance level is typically used when testing hypotheses. It represents the probability of finding a difference where none exists.

[5]And in the case of [3], also utilizing some whitebox information.

components are not particularly useful in a complex environment. Instead, we rely upon a partial factoring into a high-level entity-relationship model that reveals the bonds between services and resources. We can quantify those relationships via näive physics or via statistical inference. This gives us a relatively quick way to flag resources to change, and to check whether our intuitive ideas of what to change are valid.

Acknowledgements

We are grateful to the students of *Comp193SS: Service Science*: Hamid Palo, Ashish Datta, Matt Daum, and John Hugg, for sticking with a hard problem and coming to some surprising results without knowing just how useful their thinking would eventually turn out to be.

Author Biographies

Marc Chiarini is completing his Ph.D. at Tufts University. He previously received a B.S. and M.S. from Tufts in 1993 and an M.S. from Virginia Tech in 1998. His research interests lie in system configuration management, system behavior and performance, machine learning, and autonomic computing. Before studying at Tufts, he worked as a UNIX system administrator and toolsmith for a wide variety of companies in the San Francisco Bay area. All of his computer prowess derives from the TRS-80 COCO Introduction to BASIC manuals that he voraciously consumed at the age of nine. Marc will become a first-time father in the spring of 2009. He can be reached via electronic mail at marc.chiarini@tufts.edu .

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from M.I.T. in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Computer Science at Tufts. Prof. Couch is the author of several software systems for visualization and system administration, including Seecube (1987), Seeplex (1990), Slink (1996), Distr (1997), and Babble (2000). He can be reached by surface mail at the Department of Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as couch@cs.tufts.edu .

Bibliography

[1] Aguilera, Marcos K., Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes," *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 74-89, New York, NY, 2003.

[2] *Apache JMeter home page*, http://jakarta.apache.org/jmeter/ .

[3] Barham, Paul, Austin Donnelly, Rebecca Isaacs, and Richard Mortier, "Using Magpie for Request Extraction and Workload Modelling," *OSDI*, pp. 259-272, 2004.

[4] Brutlag, Jake D., "Aberrant Behavior Detection in Time Series for Network Monitoring," *LISA '00: Proceedings of the 14th USENIX Conference on System Administration*, pp. 139-146, Berkeley, CA, USA, 2000.

[5] Burch, Hal and Bill Cheswick, "Tracing Anonymous Packets to their Approximate Source," *LISA '00: Proceedings of the 14th USENIX Conference on System Administration*, pp. 319-327, Berkeley, CA, USA, 2000.

[6] Burgess, Mark, Hårek Haugerud, Sigmund Straumsnes, and Trond Reitan, "Measuring System Normality," *ACM Transactions on Computer Systems*, Vol. 20:, pp. 125-160, 2002.

[7] Conover, W. J., *Practical Nonparametric Statistics*, pp. 428-441, John Wiley & Sons, December, 1998.

[8] Couch, Alva, Ning Wu, and Hengky Susanto, "Toward a Cost Model of System Administration," *LISA '05: Proceedings of the 19th USENIX Conference on System Administration*, pp. 125-141, Berkeley, CA, USA, 2005.

[9] Denning, Dorothy E., "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, Vol. 13, pp. 222-232, 1987.

[10] Gunther, Neil J., *The Practical Performance Analyst: Performance-by-Design Techniques for Distributed Systems*, McGraw-Hill, Inc., New York, NY, USA, 1997.

[11] Hood, Cynthia S. and Chuanyi Ji, "Proactive Network Fault Detection," *INFOCOM*, p. 1147, IEEE Computer Society, 1997.

[12] Hoogenboom, P. and Jay Lepreau, "Computer System Performance Problem Detection Using Time Series Models," *USENIX Summer 1993 Technical Conference*, pp. 15-32, Cincinnati, OH, 1993.

[13] Jain, Raj, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley-Interscience, New York, NY, USA, 1991.

[14] Mann, H. B. and D. R. Whitney, "On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other," *Annals of Mathematical Statistics*, Vol. 18, pp. 50-60, 1947.

[15] Maxion, R. A. and F. E. Feather, "A Case Study of Ethernet Anomalies in a Distributed Computing Environment," *IEEE Transactions on Reliability*, Num. 39, pp. 433-443, 1990.

[16] Menascé, Daniel A. and Virgilio A. F. Almeida, *Capacity Planning for Web Services: Metrics, Models, and Methods*, Prentice Hall, Englewood Cliffs, NJ, USA, second edition, 2001.

[17] Menascé, Daniel A., Lawrence W. Dowdy, and Virgilio A. F. Almeida, *Performance by Design: Computer Capacity Planning By Example*, Prentice Hall PTR, 2004.

[18] Reynolds, Patrick, Janet L. Wiener, Jeffrey C. Mogul, Marcos K. Aguilera, and Amin Vahdat, "Wap5: Black-Box Performance Debugging for Wide-Area Systems," *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pp. 347-356, New York, NY, USA, 2006.

[19] Siegel, Sidney and N. John Castellan, *Non-Parametric Statistics for the Behavioral Sciences*, McGraw-Hill, NY, NY, USA, second edition, 1988.

[20] Smith, Barry and Roberto Casati, "Näive Physics: An Essay in Ontology," *Philosophical Psychology*, Vol. 7, Num. 2, pp. 225-244, 1994.

[21] Wasserman, Larry, *All of Nonparametric Statistics (Springer Texts in Statistics)*, Springer, May, 2007.

Automatic Software Fault Diagnosis by Exploiting Application Signatures

Xiaoning Ding – The Ohio State University
Hai Huang, Yaoping Ruan, and Anees Shaikh – IBM T. J. Watson Research Center
Xiaodong Zhang – The Ohio State University

ABSTRACT

Application problem diagnosis in complex enterprise environments is a challenging problem, and contributes significantly to the growth in IT management costs. While application problems have a large number of possible causes, failures due to runtime interactions with the system environment (e.g., configuration files, resource limitations, access permissions) are one of the most common categories. Troubleshooting these problems requires extensive experience and time, and is very difficult to automate.

In this paper, we propose a black-box approach that can automatically diagnose several classes of application faults using applications' runtime behaviors. These behaviors along with various system states are combined to create signatures that serve as a baseline of normal behavior. When an application fails, the faulty behavior is analyzed against the signature to identify deviations from expected behavior and likely cause.

We implement a diagnostic tool based on this approach and demonstrate its effectiveness in a number of case studies with realistic problems in widely-used applications. We also conduct a number of experiments to show that the impact of the diagnostic tool on application performance (with some modifications of platform tracing facilities), as well as storage requirements for signatures, are both reasonably low.

Introduction

Since the advent of the notion of "total cost of ownership" in the 1980s, the fact that IT operation and management costs far outstrip infrastructure costs has been well-documented. The continuing increase in IT management costs is driven to a large extent by the growing complexity of applications and the underlying infrastructure [6]. A significant portion of labor in these complex enterprise IT environments is spent on diagnosing and solving problems. While IT problems that impact business activities arise in all parts of the environment, those that involve applications are particularly challenging and time-consuming. In addition, they account for the majority of reported problems in many environments and across a variety of platforms [12].

Many factors can cause incorrect application behavior, including, for example, hardware or communication failures, software bugs, faulty application configurations, resource limitations, incorrect access controls, or misconfigured platform parameters. Although some of these are internal to applications, i.e., bugs, failures are more commonly caused when an application interacts with its runtime environment and encounters misconfigurations or other types of problems in the system [22]. Troubleshooting these problems involves analysis of problem symptoms and associated error messages or codes, followed by examination of various aspects of the system that could be the cause. Application programmers can leverage signal handlers, exceptions, and other platform support to check for and manage system errors, but it is impossible to anticipate all such failures and create suitable error indications [7]. As a result, solving these application problems requires a great deal of experience from support professionals and is often ad-hoc, hence it is very difficult to automate this process.

In this paper, we present a black-box approach to automatically diagnose several types of application faults. Our system creates a *signature* of normal application behaviors based on traces containing an extensive set of interactions between the application and the runtime environment gathered during multiple runs (or for a sufficiently long run). When an application fault occurs, we compare the resultant trace with the signature to characterize the deviation from normal behavior, and suggest possible root causes for the abnormal operation. Using output from our analysis, a system administrator or user can significantly reduce the search space for a solution to the problem, and in some cases pinpoint the root cause precisely.

We represent an application's runtime behaviors using a variety of information, including its invocation context (e.g., user id, command line options), interactions with the platform during execution (e.g., system calls, signals), and environment parameters (e.g., environment variables, `ulimit` settings, shared library versions). Our approach makes extensive use of the `ptrace` facility [8] to collect system call and related information, and other interfaces to gather additional data. Traces containing such information are created during application runtime. After observing multiple runs of an application, information from these traces are summarized (into signatures) and stored in a signature bank. If the application misbehaves, normal behavior of the application stored in the signature bank is compared with the faulty execution trace to find the root cause.

We evaluate the effectiveness of our tool using a series of real problems from three popular applications. Our case studies show that the tool is able to accurately diagnose a number of diverse problems in these applications, and its accuracy can be improved as our tool observes more traces to increase the number (and diversity) of normal execution paths reflected in the application signatures. For each of the applications we also perform detailed evaluations of the time and space overhead of our approach, in terms of the application response time degradation due to trace collection, and the storage needed to store trace data and signatures. Our initial results showed that the time overhead is very noticeable for the applications we tested, up to 77% in the worst case using standard tracing facilities. However, with some modifications and optimizations, we can reduce this to less than 6%, which is a promising indication that this tool can be used in production environment. In terms of space, we observe that signatures grow to nearly 8 MB in some cases, which is quite manageable for modern storage systems. Moreover, the space dedicated to traces and signature data can be controlled according to desired trade-offs in terms of diagnosis accuracy or application importance.

A precise definition of an application *signature* is given in the *Application Signatures* section. The *Toolset Design and Implementation* section describes the toolset we have implemented to automate the collection of trace information, construction of signatures, and analysis of faulty traces to diagnose application problems. The *Case Studies* section describe several case studies in which we apply the tool to diagnose realistic application problems in a Linux environment. The *Optimization* section includes a proposal for a technique of optimizing ptrace to significantly reduce the performance overheads incurred by trace collection. *Related Work* discusses some of related work. *Limitations* and *Conclusions and Future Work* follow.

Application Signatures

Our approach heavily relies on our ability to capture applications' various runtime behaviors (ingredients of a signature), and using which to differentiate normal behaviors from abnormal ones. These runtime behaviors can be largely captured by recording how an application interacts with the external environment. In the following sections, we describe how to capture an application's runtime behaviors and how they can be used for building a signature, which can be more easily applied for diagnosing application problems than the raw runtime behaviors.

Capturing Application Behaviors

An application interacts with its external environment through multiple interfaces. A major channel is through system calls to request hardware resources and interact with local and remote applications and services. By collecting and keeping history information on system calls, such as call parameters and return values, runtime invariants and semi-invariants can be identified.[1] Attributes that are invariant and semi-invariant are important in finding the root cause of a problem, as we will see later.

Factors that have an impact on an application's behavior can be mostly captured via information collected from system calls. However, there are some factors that can influence an application's behavior without ever being explicitly used by the application (and therefore, cannot be captured by monitoring system calls.) For example, resource limits (ulimit), access permission settings (on executables and on users), some environment variables (e.g., LD_PRELOAD), etc. cannot be observed in the system call context, but nevertheless, have important implications on applications' runtime behaviors. Additionally, asynchronous behaviors such as signal handling and multi-processing cannot be captured by monitoring system calls, and yet, they are intrinsic to an application's execution behavior. Therefore, to have a comprehensive view of an application's behavior, we collect the following information.

- **System call attributes:** we collect system call number, call parameters, return value, and error number. On a number of system calls, we also collect additional information. For example, on an open call, we make an extra stat call to get the meta-data (e.g., last modified time and file size) of the opened file. Or, on a shmat call, we make an extra shmctl call.
- **Signals:** we collect the signal number and represent information collected during signal handling separately from the synchronous part of the application. This is discussed (along with how to handle multi-process applications) in more detail later.
- **Environment variables:** we collect the name and value of all the environment variables at the application startup time by parsing the corresponding environ file in /proc.
- **Resource limits:** we collect ulimit settings and other kernel-set parameters (mostly in /proc) that might have impacts on applications.

[1]Invariants are attributes with a constant value, e.g., when an application calls readopento name of the file, given as a parameter to the call, is almost never changed. Semi-invariants are attributes with a small number of possible values, e.g., the return value of the open call normally returns any small positive integer but does not have to be a fixed number.

- **Access control:** we collect the UID and GID of the user and access permissions of the application executables.

This is not meant to be a complete list, but from our experience working in the system administration field, we believe this is a reasonable starting point and the information that we are collecting here will be useful in diagnosing most problems. In the next section, we describe how the collected information is summarized to build a signature.

Building Application Signatures

We use a simple example in Figure 1 to illustrate how signatures are constructed from application's runtime behaviors. These runtime behaviors can be broken down into their elemental form as *attributes*, e.g., an environment variable is an attribute, uid is an attribute, and each of the parameters in a system call and its return value is an attribute. Distinct values that we have seen for an attribute are stored in a set structure, which is what we called a *signature* of that attribute. For example, the environment variable $SHELL in the above example changed from "bash" to "ksh" between runs. Therefore, the signature of the $SHELL attribute is represented as a set {"bash", "ksh"}. On the other hand, the `errno` of the `open` call in the above example is always zero. Therefore, its signature is simply a set with one item {"0"}.

Some attributes always change across runs (i.e., normal runtime variants), e.g., PID, temporary file created using `mkstemp`, the return value of `gettimeofday`, etc. These are not useful attributes that we can leverage during problem diagnosis. We identify non-useful runtime variants with the one-sample Kolmogorov-Smirnov statistical test (KS-test) [5]. The KS-test is a "test of goodness of fit" in statistics and is often used to determine if values in two datasets follow same distribution. It computes a distance, called *D* statistic, between the cumulative distribution functions of the values in two datasets. The KS-test provides a critical value D_α for a given significance level α, which represents the probability that the two datasets follow same distribution but the KS-test determines they are not. If the D statistic is greater than D_α, the two datasets can be considered to have different distributions with the possibility of $1 - \alpha$. Obviously, if an attribute is a runtime invariant (i.e., only one value in its signature), we do not perform a KS-test on it.

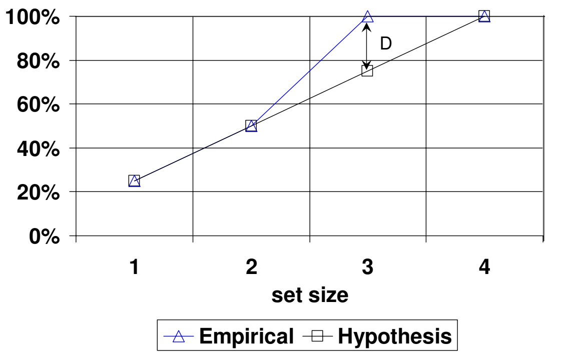


Figure 2: A simple example illustrating KS-test.

We apply the KS-test to test only attributes with more than one distinct value. For such attributes, we monitor the changes in their signature size (i.e., the set size). Thus, we have a series of set sizes (as many as the collected values) for that attribute across runs. We then hypothesize that the attribute is a runtime variant, and its value changes in each run. This hypothesis will generate another series of set sizes, and the set contains all distinct values. We then use the KS-test to determine if the distributions of the set sizes in the two series are the same. If the distributions are the same, the attribute is considered to be a runtime variant. As an example, we assume we have collected four values (and three are distinct) for an attribute. When we build its signature by merging these four values into a set one by one, we obtain 4 set sizes (1, 2, 3, 3) – the last 2 values are the same and did not increase the size of the set. If the attribute is a runtime variant, we expect

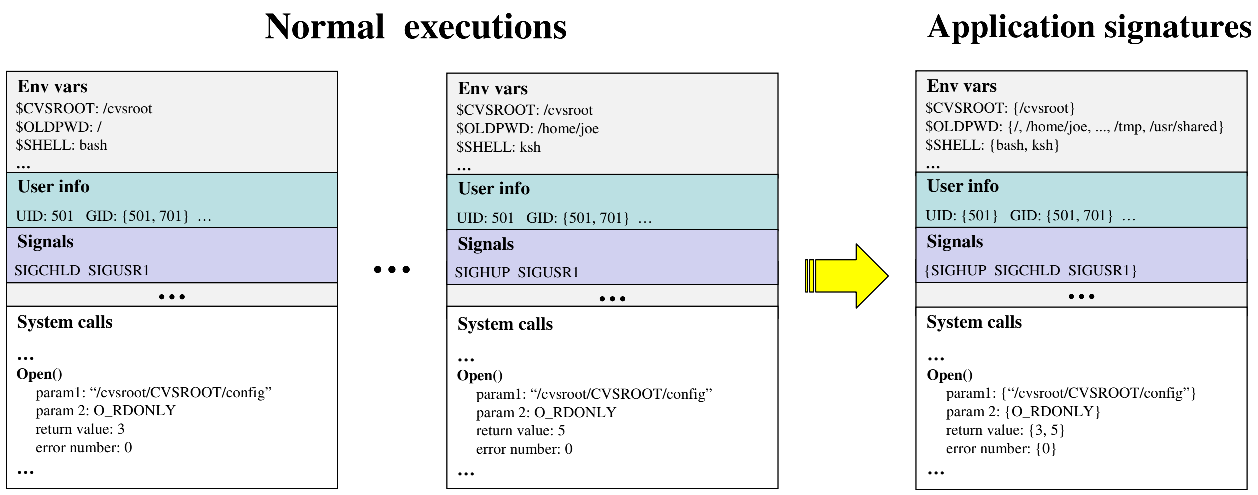


Figure 1: A simple example to show how an application signature is built from its runtime behaviors.

the set sizes are (1, 2, 3, 4). We use the KS-test to compare the cumulative distribution functions of the set sizes as shown in Figure 2. For this example, the D statistic is 0.25. If we set the significance level α to 10%, the critical value D_α of the KS-test is 0.564. As the D statistic is less than D_α, the KS-test determines that the distributions of the set sizes in the two series are the same, thus, we consider this attribute as a runtime variant.

When an application fault arises, we compare the values of the attributes collected in the faulty execution against the values in their signature. Attributes that are considered as runtime variants are not used in comparison. If a value of an attribute cannot be found in its signature, the attribute is considered to be abnormal and is identified to be a possible root cause. With the signatures built in Figure 1, if a process receives an extra signal *SIGXFSZ* in a faulty execution, which cannot be found in the signal signature, the signal can be identified to be abnormal. According to the semantics of the signal, only a process writing a file larger than the maximum allowed size receives this signal. Thus one can find the root cause by checking the size of the files used by the application. Since each file accessed is monitored, the over-sized file can be easily identified using our tool.

Building Signatures for System Calls

We have shown the method of building signatures for attributes in the previous section. However, building signatures for attributes in system calls – e.g., parameters, return value, or error number – is not as simple. Before attributes in a system call can be built into signatures, we first need to find other invocations of this system call that are also invoked from the same location within the target application, either in the same run or in a previous run. However, this is a very difficult task when trying to find these correlated system calls among hundreds of thousands of system calls that are collected.

To understand the difficulty, we use an example shown in Figure 3. In this snippet of code, there are two write calls. Either one or the other will be invoked, depending on the value of `nmsg`, but not both. It makes no sense for us to merge the attributes of the first write call with those of the second write call when generating signatures, as these two write calls perform very different functions. The first is to write messages to a file, and the second is to print an error message to `stderr`. Therefore, attributes of the first write will only be merged with those of other invocations of the first write, i.e., within the `for` loop. One can imagine how difficult it would be to differentiate the first write call from the second when looking at a trace of a flat sequence of system calls as shown in Figure 3.

We address this problem by converting a flat sequence of system calls to a graph representation, which we call a *system call graph*. Each node in the graph represents a unique system call in the target application, and the edges are uni-directional and representing the program execution flow from one system call to another. The right part of Figure 3 shows such an example graph, where the two write calls are clearly differentiated. As we can see, only those system calls invoked from the same place within the target application are collapsed into the same node, e.g., the open from the two runs and the 3 invocations of the first write call in the `for` loop from the first run. Attributes associated with each system call are appended to the node in the graph the system call corresponds to, as shown in Figure 4.

```
…
fd = open("pg.log", flag);
if( nmsg >0) {
   /* write messages */
   for(i=0; i<nmsg; i++)
      write(fd, msgs[i],
             size[i]);
}
else {
   write(2, "error!\n", 7);
}
…
```

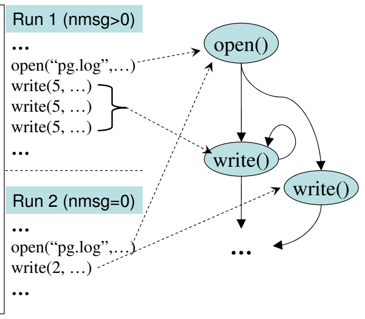


Figure 3: A sample system call graph built for the system calls shown in the middle. The program is shown on the left.

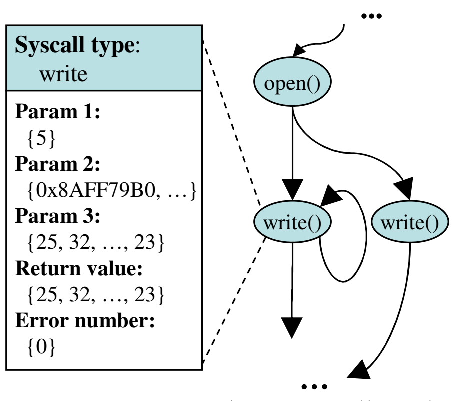


Figure 4: An example system call graph.

A vital step in the construction of the system call graph is to collapse system call invocations that are invoked from the same location in the program to a single node in the graph, either within a single run or across multiple runs. Though the locations in an application can be represented by their virtual memory addresses, we use the stack of return addresses by collecting and analyzing the call stack information of the target process during each system call invocation. This gives system calls an *invocation context* in a more accurate way. The program in Figure 5 illustrates this point. For the statements in the program, their memory addresses are shown on the left side of them. In the program, open and write system calls are wrapped in low level functions openfile and writestr. As these functions are used in different places in the program for different purposes, the system calls wrapped in them are also invoked for different purposes. Take function openfile as an example. It is used in two places in the program. In one place, it is to open an log file, and in the other place, it is to open a temporary file. Thus open is indirectly called two times for two different purposes. It is necessary to differentiate the open invocations for opening the log file and invocations for opening a temporary file because the names of temporary files are randomly generated and change across different runs. To clearly differentiate these two types of open invocations, we need not only the address where open is called, but also the addresses where openfile is called. This example illustrates that a "stack" of addresses of the functions in the calling hierarchy are needed to accurately differentiate the system call invocations.

We show the algorithm for collapsing a flat sequence of system call invocations to a system call graph in Figure 6. On line 6, the algorithm searches a matching node for a system call invocation following the edges in the system call graph. On line 14, when a node in the system call graph is found for the system call invocation, the attributes of the invocation, e.g., parameters, return values, and error numbers, are merged with existing attributes of the node. For each system call attribute, we again use a set to represent its distinct values among different invocations. This is illustrated in a write node of Figure 4.

Dealing with Multiple Processes

Applications, especially server applications, may have multiple processes running concurrently. We collect data for each process separately for two reasons. The first reason is that the causal relations between system calls can only be correctly reflected after separating interleaving system calls. Both building system call graphs and diagnosis require to know correct causal relations between system calls. While building system call graphs needs the causal relations to form

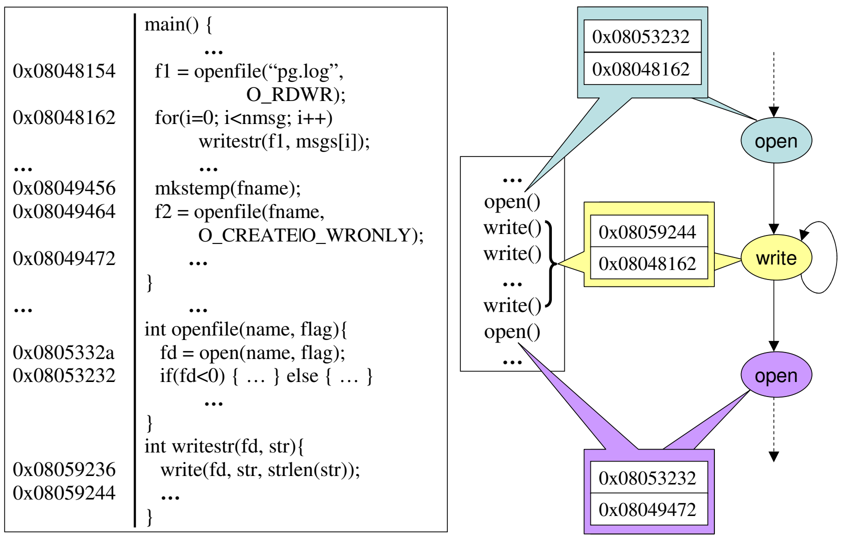

```
                 main() {
                     ...
0x08048154         f1 = openfile("pg.log",
                                 O_RDWR);
0x08048162         for(i=0; i<nmsg; i++)
                         writestr(f1, msgs[i]);
...                      ...
0x08049456         mkstemp(fname);
0x08049464         f2 = openfile(fname,
                         O_CREATE|O_WRONLY);
0x08049472               ...
                 }
...                      ...
                 int openfile(name, flag){
0x0805332a          fd = open(name, flag);
0x08053232          if(fd<0) { … } else { … }
                         ...
                 }
                 int writestr(fd, str){
0x08059236          write(fd, str, strlen(str));
0x08059244          ...
                 }
```

Figure 5: An example shows building system call graph with return addresses in call stack.

```
 1:  prev_node = NULL
 2:  for each system call in the flat sequence do
 3:      if the graph is empty
 4:          curr_node = NULL
 5:      else
 6:          search the nodes pointed by edges starting on prev_node using context
 7:          set curr_node as the node having the same context or NULL
 8:      endif
 9:      if curr_node = NULL
10:          add a new node (referred as curr_node)
11:          populate system call attributes in curr_node
12:          add an edge from prev_node to curr_node
13:      else
14:          update system call attributes in curr_node
15:          add an edge from prev_node to curr_node
16:      endif
17:      prev_node = curr_node
18:  end
```

Figure 6: Algorithm to convert a system call sequence to a system call graph.

correct paths, diagnosis requires causal relations to trace back to system calls ahead of the anomalies to get more information. For example, if we identify that a write call is an anomaly, we want to get the pathname of the file it changes by tracing back to an open call with the file descriptor. The second reason is that some attributes like signals, UIDs and GIDs are specific to a process. It is necessary to collect their values in a per-process mode to build accurate signatures for these attributes.

When we build signatures for a multi-process application, we divide its processes into groups based on the roles they play in the application, and build signatures separately for each process group. For example, a PostgreSQL server may create one or more back-end processes, one daemon process, and one background writer in each run. We build a system call graph and form a set of signatures for back-end processes, and we do the same for the daemon processes and background writers. When we build signatures for each process group, we treat the data collected for a process just like that collected in an execution of a single process application, and build signatures in a similar way. To identify which group a process belongs to, we use the stack information (return addresses) of the system call creating the process as a *context* of the process. Processes with same *context* are considered to be in the same group.

For multithreaded applications, we collect data and build system call graphs and signatures for threads in the same way as we do for processes by treating each thread just like a process. While we can differentiate native threads through ptrace and /proc interfaces, which are managed by OS kernel, we cannot differentiate user-level (green) threads, which are managed at user space and thus transparent to OS kernel. As user-level threads have not been widely used, our current approach does not handle user-level threads.

We handle signal handler functions similarly to child processes, except that we collect only signal number and system call attributes for signal handler functions. When we build signatures for signal handlers, we use a 2-element tuple <process context, signal number> as the *context* of a signal handler. Thus, only data collected for signal handlers that handle same type signals for processes with same context are summarized to form signatures, e.g., we build a set of signatures for the SIGHUP signal handlers in the back-end processes of PostgreSQL.

Toolset Design and Implementation

In this section, we describe the architectural design and implementation of our diagnostic toolset for capturing applications' runtime behaviors, building signatures, and using which to find root cause of problems when they arise.

Figure 7 shows the overall architecture of our diagnostic toolset. First, a *tracer* tool (in the *Application Tracer* subsection) is used to monitor the runtime behaviors of applications and record a log of these behaviors. Logging is started by having the tracer tool fork-execute the target application, e.g., *'tracer sample_program'*. However, the tracer tool can be used more seamlessly if we attach it to the shell process and have it monitor all of the child processes created by the shell. Since this tool is intended to run alongside of applications at runtime, having low overheads is crucial. We will see a detailed study of time and space overheads in the *Case Studies* section.

On each run of the target application, the tracer tool will record and summarize its runtime behavior into a trace. Multiple traces are then aggregated into a *signature bank*, a central repository where the target application's runtime signatures are distilled and built. We give an in-depth explanation of the steps involved in building runtime signatures in the *Signature Bank* subsection.

The last part of the toolset, called the *classifier* (in the *Fault Diagnosis* subsection), is used when an application is misbehaving. It is used for comparing the faulty execution trace (collected by the tracer) with the application's signature bank and classifying what differing features of the faulty trace from those in the

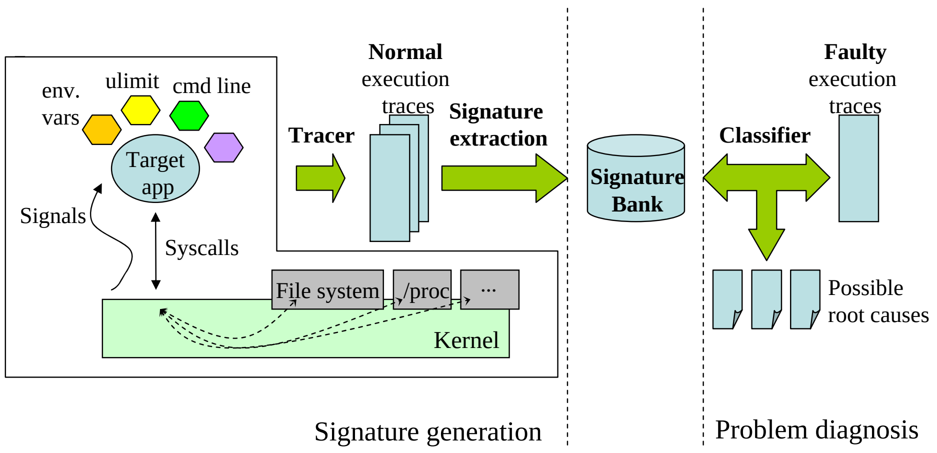


Figure 7: System architecture.

signature bank might be the root cause of the problem. It is possible that sometimes multiple differing features are found. Since there is usually only one root cause, others are false positives. In the *Accuracy and Effectiveness* section, we discuss how to reduce the number of reported false positives.

Application Tracer

The tracer tool monitors an application's runtime behaviors via the ptrace interface, which is widely implemented on Linux and most UNIX variants. This approach has the benefit of not requiring instrumenting the target application or having access to its source code, and also does not need kernel modifications. Each time the target application invokes or finishes a system call or receives a signal, the application process is suspended and the tracer is notified of the event by the kernel and collects related information, e.g., call number, parameters, and return value. For a small set of system calls, we also collect some additional information that might be useful during problem diagnosis. This information is collected usually by having the tracer make extra system calls. For example, when open is called on a file, we make an extra stat call on the opened file to get its last modified time, which will become a part of the information we collect for that open call. In addition to files, we also collect additional information for other system objects such as shared memory, semaphore, sockets, etc. As explained in the *Building Signatures for System Calls* section, to construct a system call graph from a sequence of system calls, the tracer also takes a snapshot of the call stack of the target application in the context of each system call.

As mentioned in the *Capturing Application Behaviors* section, not all runtime behaviors can be captured by monitoring system calls, e.g., environment variables, ulimit, uid/gid of the user, etc. These information are collectively obtained by the tracer at the startup time of the target application, and they may be updated by monitoring system calls such as setrlimit, setuid, etc. at runtime.

For a single-process application, tracer puts all the monitored data into a single trace file. The trace file is logically separately into multiple sections to hold different categories of runtime data, similarly to that shown in Figure 1. The largest section by far is usually the system call section. To reduce space overheads, instead of saving a flat sequence of system calls, we convert it into a system call graph on-the-fly using the algorithm described in Figure 6. The conversion removes much redundant information by collapsing multiple system calls invoked in a loop into the same system call graph node. To reduce I/O overheads, the system call graph is kept in tracer's memory space via memory mapping of the trace file.

For a multi-process application, we keep one trace file per process (by detecting fork/exec), so we can separate the interleaving system calls made by different processes and maintain process-specific state information in each trace file. Ancestry relationships between processes are also kept in the trace file so we know exactly how the trace files are related and also at which point in the parent process the child process is spawned. Signal handlers are handled the same way by the tracer, as ptrace can also trap signals.

If a long-running application has large variations in its execution, its trace files may be filled with large volumes of data collected for runtime variants. By not saving these data into traces, we can reduce space overhead without influencing diagnosis. For a attribute having been considered as a runtime variant, we set an upper limit (512 in our current implementation) on the size of the set holding its distinct values. Thus new values of a runtime variant are not collected into traces or merged into signature bank when the set size

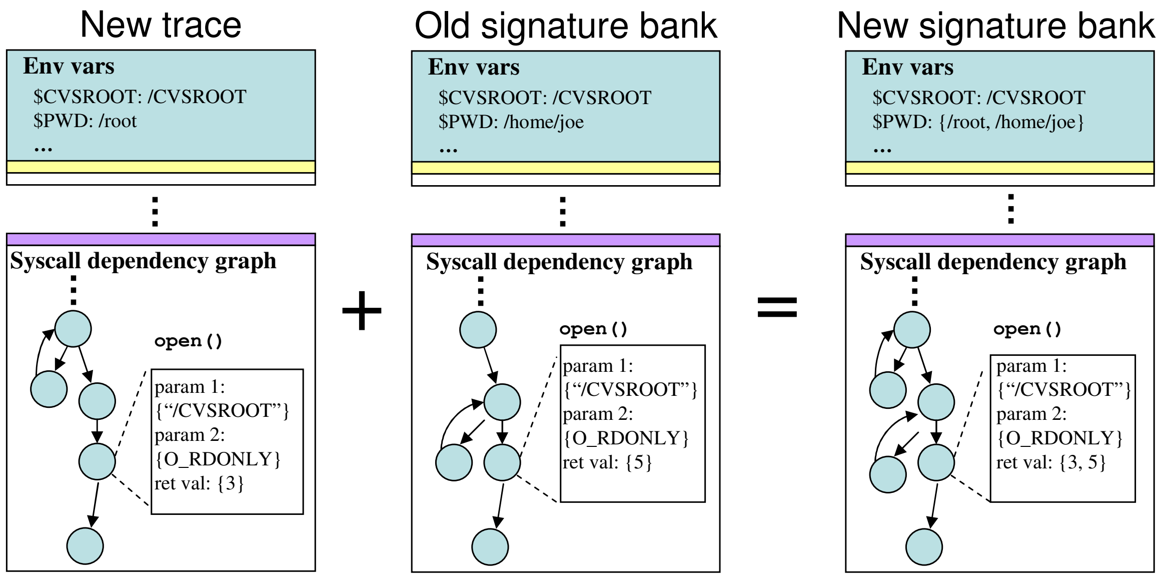


Figure 8: An example showing how a trace file is merged with the signature bank.

reaches 512. 512 is chosen so it is sufficient to cover semi-invariants with large number of distinct values, yet small enough for it not to be a storage burden.

Signature Bank

For single-process applications, a signature bank is simply an agglomerate of one or more normal execution trace files. When adding the first trace file to an empty signature bank, the trace file simply becomes the signature bank. As illustrated in Figure 8, when adding new trace files to the signature bank, values of attributes (e.g., an environment variable) in theses traces are compared to those in the signature bank. If the value of an attribute in the new trace is different from that of the signature bank, the new value is added to the set of possible values of that attribute in the signature bank. Otherwise, the signature bank remains unchanged. Since most attributes do not change between runs, the size of a signature bank grows very slowly over time. When merging the system call graph in a trace file into the signature bank, we use a similar algorithm as that described in Figure 6.

All attribute values and system call graph paths are versioned in the signature bank. This is useful when a faulty execution trace is inadvertently added to the signature bank. Versioning allows this action to be easily reverted.

For a multi-process application, its signature bank is consisted of multiple sub-banks, each of which describing a separate process group. These sub-banks are organized to reflect the ancestry relationships between the processes they are associated with. Merging of the trace files of a multi-process application into the sub-banks is performed following the algorithm described in Figure 9.

Our current approach has to rebuild application signatures after some administrative changes. For example, updating the application or the shared libraries changes the return addresses of the functions which invoke system calls directly or indirectly. Because these addresses are used as context to build system call graphs and to match system call invocations, system call graphs and signatures built before an update cannot be used any more after the update because we cannot find signatures correctly with the return addresses in new context. Thus the signature bank needs to be rebuilt to reflect the changes.

Fault Diagnosis

When an application fault occurs, a *classifier* tool is used to compare the faulty execution trace with the application's signature bank. The comparison is straight-forward. Application and system states in the faulty execution trace are first compared with those in the signature bank. Mismatched attributes are then identified. The system call graph in the faulty execution is next compared with that in the signature bank, a node at a time. For each node, its attributes are compared with those on the corresponding node within the signature bank. We do not list all the mismatched attributes as potential root causes – this might result in too many false positives.

To highlight the more likely root causes to the person diagnosing the problem, the classifier ranks the results. If an attribute from the faulty execution mismatches a signature that is either an invariant or has a very small cardinality, it is more like to be the root cause than if the signature were to a higher cardinality value. Additionally, among the mismatched attributes found in a system call graph, we give more weight to those attributes located closer to the "head" of the graph. The reason being, due to causal relationship, the mismatched attributes that are closer to the top of the call graph are likely to be the cause of the mistakes found toward the bottom.

Case Studies

In this section, we evaluate our approach using real-world application problems. We would like to

```
FUNCTION aggregate(trace_file, sub_bank)
    add trace_file into sub_bank;
    FOR each child process DO
         let trace_file_child be its trace file
         look for a child of of sub_bank using the context
                of the child process (referred as sub_bank_child);
         IF such child does not exist
              create a new sub_bank (referred as sub_bank_child);
              make sub_bank_child a child of sub_bank;
         END IF
        aggregate(trace_file_child, sub_bank_child);
    END DO
END FUNCTION

IF signature_bank is empty
    create a new sub_bank (referred as sub_bank_root);
END IF
trace_file_root = trace file of the main process;
sub_bank_root = root of the sub_bank tree;
aggregate(trace_file_root, sub_bank_root);
```

Figure 9: Algorithm to aggregate trace files into signature bank for a multi-process application.

observe how effectively and accurately the tool is able to handle these problems and also identify some of its limitations.

Experimental Methodology

Our evaluation covers three popular applications: Apache web server [1], CVS version control system [15], and PostgreSQL DBMS server [14]. Rather than injecting contrived faults to test our system, we evaluated actual problems faced by users of these applications, drawn from problem reports on Internet forums and from bug reporting tools such as Bugzilla. Our target problems include configuration files, environment variables, resource limitations, user identities, and log files. Software bug detection is not our goal in this work. We describe a subset of our experiments in this section, with the representative problems shown in Table 1.

For each application, our general approach was to first collect traces by running it with a series of standard operations or workloads that represent its normal usage and operation. In some cases, we also change some system settings to emulate administrators tuning the system or modifying configurations. For example, when collecting traces for CVS, we perform the commonly used CVS operations such as import, add, commit, checkout etc. multiple times on different modules in both local and remote CVS repositories. The CVS repositories are changed by resetting shell environment variable $CVSROOT. We integrate these normal operation traces into the signature bank to generate the runtime signatures of the application. After these two steps, we inject the selected fault manually and collect the faulty execution trace for each problem scenario. Afterward, the system is returned to the non-faulty state. Finally, we use the classifier to identify possible root causes by comparing the faulty execution traces with the application's signatures.

In each case we discuss the ability of the classifier to effectively distinguish erroneous traces from normal signatures to aid in diagnosing the problem. In addition, since the applications being diagnosed (and their threads) must be launched from our tracer tool, the performance impact as well as space overhead due to trace and signature storage are important measures of the feasibility of our diagnosis approach. Therefore, for each application we estimate overhead in execution time or response time slow down by repeating the execution without tracer. We also record the size of the individual traces and the signature bank. Trace file size is less important than the size of the signature bank since trace files can be deleted after they are inserted into the signature bank. However, if the size

Apache Problems		
Index	Symptom	Root Cause
1	Intermittent failures of httpd processes	Log file size is getting too large, close to 2 GB
2	Httpd cannot start	File system containing httpd log files is mounted as read-only
3	Httpd cannot start, because it is unable to load share libraries in correct version.	Paths are re-ordered in $LD_LIBRARY_PATH.
4	Httpd crashes when the number of connections is large.	Httpd loaded by crond has more restrictive resource limit.
5	Web clients cannot access contents pointed by symbolic links	User removed the *FollowSymLinks* directive from httpd.conf
CVS Problems		
Index	Symptom	Root Cause
6	User cannot check out a specified CVS module	$CVSROOT is pointing to a directory that is not a CVS repository
7	CVS client cannot connect to CVS server	A non-default ssh port number is specified in /etc/ssh/ssh_config
8	Accesses to CVS repository are denied	User is not added to CVS group
9	User cannot connect to CVS server with an error message "temporary failure in name resolution"	Network cable is disconnected
PostgreSQL Problems		
Index	Symptom	Root Cause
10	DBMS accepts only connections from local machine	Config file pg_hba.conf is mistakenly changed
11	Server cannot start	A stale postmaster.pid file is left undeleted after improperly shutting down the server

Table 1: Description of the problem symptoms and their root causes for Apache, CVS, and PostgreSQL.

of trace files is reasonably small, we can retain several recent traces and batch the aggregation operation to amortize the cost of insertion into the signature bank.

All the experiments below are performed on a Dell Dimension 3100 desktop computer with a 3 GHz Intel Pentium 4 CPU and 1 GB memory running Red Hat Enterprise Linux WS v4 and Linux kernel version 2.6.9.

Apache

For tests with Apache, we use WebStone 2.5 [19] to emulate multiple clients which concurrently access web pages through Apache. Besides generating workloads, we also use WebStone to measure the average response time of Apache. The web pages served by Apache are generated by LXR [13] (Linux Cross-Reference), which is a widely used source code indexer and cross-referencer. We use LXR to serve user queries for searching, browsing, or comparing source code trees of three versions of Linux kernels.

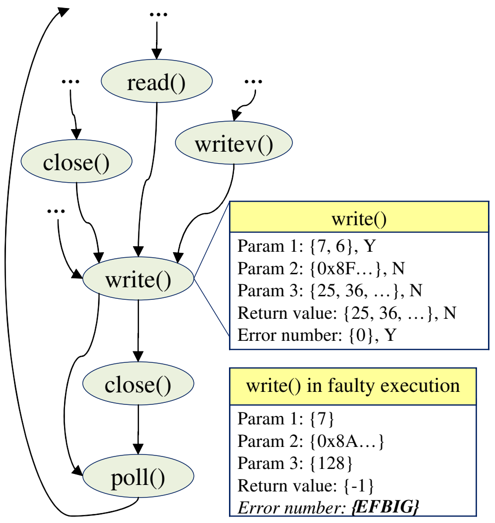


Figure 10: Signatures of the attributes in a write system call and the values of these attributes in faulty execution in problem 1. The "Y" or "N" after each signature (set) indicates whether the signature is an invariant or a semi-invariant that passes KS-test and thus can be used for diagnosis. The abnormal attribute in faulty execution is in italic font.

We repeat the following operations ten times to generate ten corresponding traces of Apache server: start the server with tracer, run WebStone on another machine for 45 minutes generating HTTP requests, and stop the server.

We use the signature bank built from the traces to diagnose the Apache problems listed in Table 1. Both problems 1 and problem 2 are related to log files. Because the contents and the sizes of log files usually change frequently, problems related to log files are difficult to diagnose by directly comparing persistent states without capturing the run-time interactions of the application. Our classifier identifies the root causes by finding out abnormal system calls in the faulty execution traces, write for problem 1 and open for problem 2. The abnormally behaved system calls are identified because their error numbers do not match their signatures captured in the signature bank. Figure 10 illustrates the difference between the values of these attributes in the faulty execution and their signatures in signature bank for problem 1. In problem 1, system call write in faulty execution cannot write access logs into log file access_log successfully. The root cause is revealed from its error number (*EFBIG*, which means file is too large). Similarly, in problem 2, system call open in faulty execution cannot open file error_log successfully. The root cause is revealed from the return value(-1, which means the system call fails) and its error number (*EROFS*, which means read-only filesystem). In addition to abnormally behaved system calls, the classifier also identifies that some httpd processes receive *SIGXFSZ* signals in the faulty execution in problem 1. The *SIGXFSZ* signal is only thrown by the kernel when a file grows larger than the maximum allowed size.

Figure 11 illustrates the command used and the output of our tool in diagnosing problem 1. The classifier command usually has two parameters, the faulty execution trace ("traces/Apache_problem1.trace"), and the signature bank of Apache (stored in a file named "sigbank/Apache"). The messages under the command are console output of the classifier. The first line of the console output shows one of the possible root causes of this problem – the abnormal write system call invocation. Record_ID, Node_ID and Graph_ID indicate where the signatures are located in the signature bank so users can manually check the entire system call graph if necessary. The second and the third lines show how the system call invocation behaves abnormally. The remainder of the console output reports a second possible root cause, namely the new signals which don't appear in normal executions.

Problem 3 of Apache is caused by a modified environment variable. The classifier identifies the environment variable ($LD_LIBRARY_PATH) by

```
[sigexp@sysprof ~]$ classifier sigbank/Apache traces/Apache_problem1.trace
** Record_ID: 58        Node_ID: 58     Graph_ID: 6     System call: write
Fails to write file /m/logs/access_log.
Note: File too large.
** Signal SIGXFSZ received by process 8259, 8260, 8261, 8262, 8267, 8268, 8269, 8271
Signal appears only in faulty execution.
Note: application appends a file larger than maximum allowed size.
```

Figure 11: Command line and console output of Classifier diagnosing Apache problem #1.

comparing the value of the environment variable in the faulty execution trace against those in the signature. When Apache performs normally, paths in the environment variable are in the right order, and Apache can load correct libraries. Since this variable usually does not change in normal executions, we capture the value of $LD_LIBRARY_PATH as a signature in the signature bank. In the fault execution, paths in $LD_LIBRARY_PATH are reordered. As a result, when comparing the faulty execution trace against the signatures, the classifier finds the new value does not match the value in the signature bank and reports it as a possible root cause. Besides the changed environment variable, the classifier further identifies that the fault is caused by opening incorrect files in faulty execution because the pathnames of these files are different with those in the signatures. Based on the pathnames, administrators may identify these files are shared libraries.

Problem 4 is caused by a restricted resource limit setting on the maximum number of processes owned by the same user. Our classifier diagnoses this problem by observing the abnormal return values and error numbers of the setuid system calls made by the httpd processes. The setuid system call increases the number of processes owned by the user which Apache runs as. The return values indicate that the system calls did not succeed, and error numbers indicate that the failure was caused by unavailable resources. In addition, since we keep resource limit as an attribute of the shell environment signature. The new resource limit value in the faulty execution differs with that in the signature, which is another indication of the root cause.

Problem 5 is caused by a change in a config file httpd.conf. In building application signatures, file metadata such as file size, last modification time are collected, usually when an open call happens. When comparing the faulty execution trace to the signatures, our classifier discovered that attributes of httpd.conf such as file size, last modification time etc. do not match those in the signatures. Thus our classifier can attribute the application failure to the change in httpd.conf.

In these experiments, the response time of Apache observed by WebStone is increased by 22.3% on average. The performance overheads are non-negligible. We propose a method to reduce performance overheads in the *Optimization* section.

Figure 12 shows the change in size of an Apache trace in a 45 minutes period when Apache is serving requests. In the first a few minutes, the system call graphs are small and the value sets for the attributes do not include so many distinct values. The trace grows quickly as new system call graph nodes and new values are added into the trace. Afterward, the growth slows down with the system call graphs becoming more and more complete and the value sets covering more variations of the attributes. This trend is apparent especially after the 30th minute due to redundancy across requests. At the end of the execution, the trace occupies 6.3 MB space, recording nearly 11 million system call invocations.

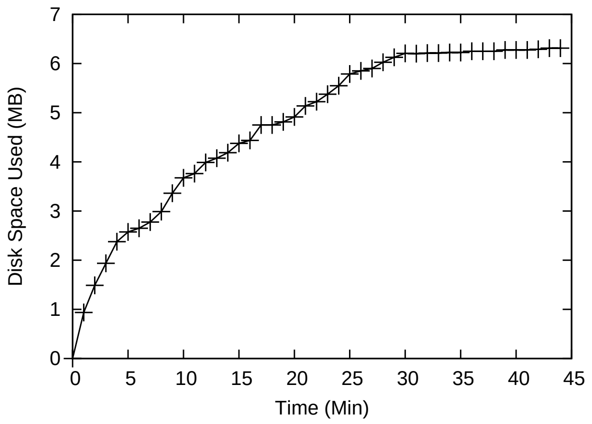


Figure 12: The size change of an Apache trace.

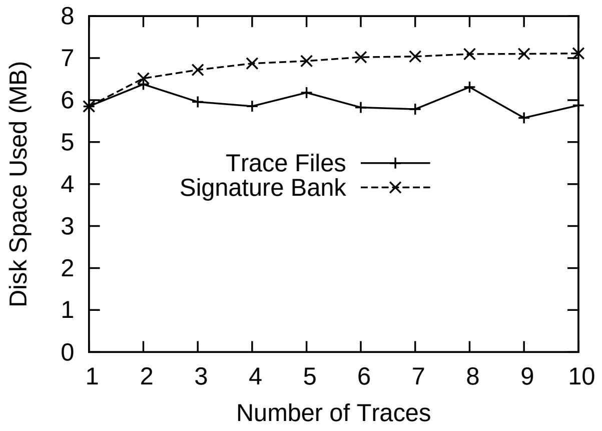


Figure 13: Sizes of traces and the signature bank for Apache.

Figure 13 shows the size of the traces, and the change in the size of the signature bank after aggregating each of the traces. Though the size of each trace is around 6 MB, the size of the signature bank grows very slowly when a new trace is inserted because redundant data are merged.

CVS

As we have explained, we collect traces of commonly used CVS operations on different modules including the source code of our diagnostic tools, strace, Gnuplot, and PostgreSQL in both local and remote CVS repositories.

Similar to problem 3, problem 6 is also caused by a modified environment variable. The symptom of problem 6 is that an user cannot check out a specified CVS module. Figure 14 illustrates the command used and the console output of our tool. The first line of the console output shows one of the possible root causes of this problem – a new $CVSROOT's value has been used in the faulty execution. When CVS performs normally, the environment variable $CVSROOT has been changed several times and pointed to different repositories. These repositories include a local one at /home/cvs/repository and several remote ones, from which we checked out the source code of strace, Gnuplot and PostgreSQL. Though this variable has been changed multiple times, our tool determines with a KS-test that this attribute is not a runtime variant, and uses its signature in diagnosis because the D-statistic of this variable is 0.42, which is far above the corresponding critical value $D_\alpha = 0.22$. In the faulty execution, $CVSROOT is changed to /home/cvs. As a result, the classifier finds the new value does not match the signature and reports it as a possible root cause.

Our tool also discovers (lines 2-4 of the output in Figure 14) an abnormal access system call invocation. The access system call is made by CVS to check the access permission of the CVSROOT directory. In normal executions, the "pathname" parameter is "/home/cvs/repository/CVSROOT", the return value is 0, and error number is 0. However, in the faulty execution, the access call has a different "pathname" parameter ("/home/cvs/CVSROOT") because $CVSROOT has been changed to /home/cvs. /home/cvs/CVSROOT is a non-existent directory. Thus the system call returns -1 and the error number is set to ENOENT accordingly. Our classifier interprets the semantics of the return value and error number so users can understand easily.

This simple example demonstrates how our tool helps to pinpoint the root cause of the problem and reveal detailed information for users to examine and verify, while the error message printed by CVS is simply "cannot find module 'strace' – ignored", which is not very descriptive and may be misleading.

Problem 7 is about a failed CVS server connection because of a non-default SSH port number in the configuration file. CVS usually makes connections with the remote CVS server via SSH using its default port number (number 22). In this scenario, the configuration file of the SSH client, /etc/ssh/ssh_config, has been modified to use a customized port number. Therefore, all SSH client requests will use this customized number instead of the default port number. However, the SSH server on the CVS server is not changed accordingly to accept this new port. Our tool identifies the config file to be one of the root causes in a similar way as in problem 5. When comparing the faulty execution trace to the signatures, our classifier discovers that the file was modified when the application is doing an open call, since the file size, last modification time etc. do not match. Beside the config file, our classifier also reports that a connect system call invocation is having a different port number as its parameter. This information indicates the cause might be a bad port number.

Problem 8 is one of the problems used to evaluate AutoBash [17], we revisit this problem with our approach. AutoBash solves this problem by looking for the causality between the group identifiers (*gids*) of the user and the access permissions of CVS repository. Our approach builds a signature for *gids* used in CVS normal executions. In our signature bank, the signature of this attribute always takes one value since the CVS client always uses the CVS group. When comparing the faulty execution trace against the signatures, the classifier cannot find the *gid* of CVS group in the set of *gids* used by the faulty execution, thus it classifies it as the root cause. Similar to problem 6 and problem 7, the classifier observes abnormally behaved system calls in faulty execution trace and prints out diagnosis messages of the errors.

From the problems we present here, the only problem for which the classifier cannot provide accurate diagnosis is problem 9. The classifier observes the abnormal behavior of the poll system call recorded in the faulty execution trace and concludes that poll gets a timeout as the root cause. The classifier fails to identify the real root cause, because we do not collect information about hardware states of the network card. Though the classifier cannot exactly locate the root cause, it discovers that the anomaly was caused by timeout on network communications. The information may be helpful because it can reduce the scope of investigation for the exact root cause.

While the tracer slows down CVS operations by different percentages, we observe an average slowdown of 29.6%. The smallest slowdown is less than 1%. It is observed when we check out Gnuplot from the remote repository gnuplot.cvs.sourceforge.net because network latencies dominate the delays. The greatest slow-down is 77.1%, which is observed when we commit a version of a small module to the local repository. We collected 26 traces for CVS in total. Their sizes range from 0.1 MB to 1.6 MB. They record about 1.8 millions system call invocations, and the largest trace file records over 219 thousands system call invocations. The size of the signature bank is 6.5 MB after these traces are aggregated.

PostgreSQL

For PostgreSQL, we collected 16 traces as it processed queries generated by the TPC-H [18] benchmark for decision support systems.

```
[sigexp@sysprof ~]$ classifier sigbank/CVS traces/CVS_problem6.trace
Environment variable $CVSROOT has been changed to a new value "/home/cvs".
** Record_ID: 158       Node_ID: 95     Graph_ID: 1     System call: access
Faulty execution checks user permission of a file/directory "/home/cvs/CVSROOT".
System call fails.
Note: No such file or directory.
```

Figure 14: Command line and console output of Classifier diagnosing CVS problem #6.

In PostgreSQL, access control configurations are specified in pg_hba.conf. PostgreSQL loads this config file when it is started, and also does a reload when receiving a SIGHUP signal. Thus with a reload command which sends PostgreSQL a SIGHUP signal, users may make the changes to pg_hba.conf take effect immediately without restarting PostgreSQL. In evaluating problem 10, we injected faults by modifying pg_hba.conf when PostgreSQL was running and let PostgreSQL reload pg_hba.conf with the reload command. We run reload commands to let PostgreSQL load pg_hba.conf in its signal handler, which we have exercised in normal execution. Our classifier identifies the root cause in a similar way as it did when diagnosing problem 5 and problem 7. The console output is shown in Figure 15.

In problem 11, the shell script which loads PostgreSQL checks for the existence of the postmaster.pid file. If the file exists, it stops loading PostgreSQL, assuming it has been started already. In normal executions, an access system call is used to check for the existence of this postmaster.pid file, and usually returns -1 with the error number set to *ENOENT*. In faulty execution, the system call returns 0, indicating the existence of the file. Our classifier discovers the root cause by comparing the error numbers and return values of the access call.

We observed that, using the tracer, the queries are slowed down by 15.7% on average. Tracing causes less performance overhead for PostgreSQL than for the other two applications because most TPC-H queries are computation-intensive, and thus PostgreSQL makes system calls infrequently. The traces are from 0.6 MB to 2.1 MB, and the signature bank is 3.2 MB after aggregating the traces.

Accuracy and Effectiveness

Our approach identifies root causes of problems by comparing a faulty execution with the application's normal runtime signatures. Having "good-quality" runtime signatures is critical to the identification of root causes. From our experience, identifying the root cause is usually not difficult using our approach, as we are comprehensively capturing the interactions between the application and the system states, whether or not they are persistent or non-persistent (the root causes of the above problems are all correctly identified using our tool). In addition to being able to identify root causes, it is also important, if not more important, to limit the number of false positives. Having too many false positives will render the tool useless in practice.

False positives are generally caused by two reasons. One reason is related to the KS-test. Some normal runtime variants may not be ruled out during diagnosis if the significance level is set too high. An user may reduce false positives by decreasing the significance level. However, if the level is set too low, attributes useful for diagnosis may be mistakenly identified as runtime variants and thus lead to false negatives. From our experience, setting the level to 10% works well for all the problems in our experiments (the numbers of false positives in diagnosing the problems are as shown in Figure 16). Nevertheless, we have enabled the significance level to be set as a knob, in case users may need to adjust it in real-world environments to reduce false positives without causing false negatives.

The other reason is that signature bank cannot cover all the possible normal variations of the attributes. For example, in problem 6, if the client has never connected to a CVS server before, the signature of $CVSROOT does not include the name of the new repository. Thus the name of that new repository in $CVSROOT may be identified as one of the possible root causes false-positively. Aggregating more traces may make signature bank more "complete," and thus is helpful in reducing such false positives. To illustrate this, for each problem, we also show the number of false positives in Figure 16 when we increase the number traces aggregated into the signature bank.

```
[sigexp@sysprof ~]$ classifier sigbank/postgresql traces/postgresql_problem10.trace
** Record_ID: 45287     Node_ID: 11     Graph_ID(SIGHUP): 3     System call: open
File /home/pgsql/db/pg_hba.conf has been changed since last run.
```

Figure 15: Command line and console output of Classifier diagnosing PostgreSQL problem #10.

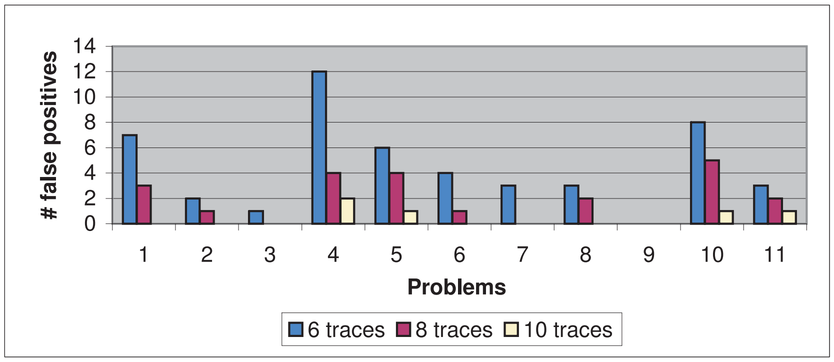


Figure 16: Number of false positives decreases when more traces are aggregated into signature bank.

Optimization

Our experiments in *Case Studies* show that the performance overheads of tracing are quite noticeable when using on real systems. In this section, we propose a technique of optimizing ptrace to significantly reduce these overheads.

Most of the performance degradation comes from information collection and trace file updating when a system call happens. To reduce the context switches and memory copies introduced by updating trace files, we have used direct memory-mapping to map trace files into the memory space of the tracer. However, for each each system call made by the traced application, the following overheads are still incurred.

- Four additional context switches, switching from kernel to tracer and back from tracer to kernel both at system call entry and exit. Time consumption is about 20.2 microseconds in total.
- Getting system call number, return value, error number, or each parameter would incur two additional context switches of 0.9 microseconds.
- Peeking into the user stack of the target application to get the content of its stack frames would require the OS to read the application's page table to resolve virtual addresses. Each of these operations takes about 2.0 microseconds.

Since most system calls usually take only a fraction of a microsecond, in the same time scale or even shorter than these activities, these overheads may significantly slow down the traced application. To reduce these overheads, we modified several ptrace primitives and added two primitives in Linux kernel. These improvements only require slight modifications to the current ptrace implementation. Less than 300 lines of new code are added. The new ptrace actions/primitives we added are:

- PTRACE_SETBATCHSIZE: Set the number of system calls to batch before notifying the tracer.
- PTRACE_READBUFFER: Read and then remove data collected for the system calls in same batch from a reserved buffer space.

The improved ptrace interface reduces overheads by decreasing the number of ptrace system calls the tracer needs to call and the number of context switches. This is done by having the kernel reserve a small amount of buffer space for each traced process (40KB in the current implementation) so it can be used by ptrace to store data it has collected on behalf of tracer without interrupting the traced application on every system call. Instead, the traced application is only interrupted when (i) the buffer space is approaching full, (ii) a user-defined batch size (of system calls) is reached, or (iii) a critical system call is made, e.g., fork, clone, and exit. By batching the collection of information on system calls, the costs of context switches and the additional ptrace system calls are dramatically reduced.

We repeated the trace collection operations for *Apache*, *CVS*, and *PostgreSQL* in the *Case Studies* section with the improvements introduced above. The slowdowns of these applications are shown in Figure 17 with the batch size varying from 1 to 64.

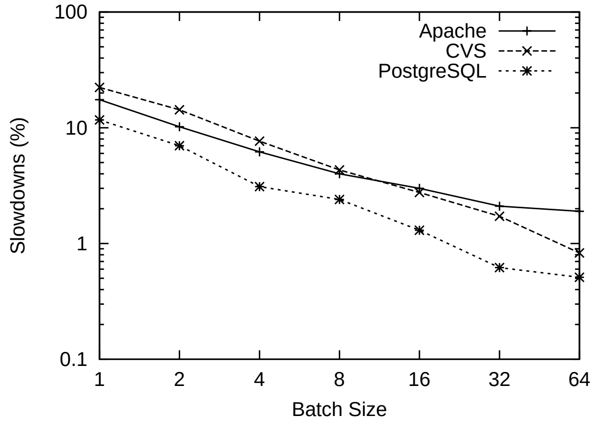


Figure 17: Slowdowns for *Apache*, *CVS*, and *PostgreSQL*, with batch size varying from 1 to 64.

Even when batch size is equal to 1, the applications have smaller slowdowns with improved ptrace than they do with original ptrace. There are two reason. One reason is that OS invokes tracer only once with improved ptrace for each system call on its exit, instead of twice with original ptrace on both system call entry and system call exit. The other reason is that the tracer needs only one improved ptrace system call (PTRACE_READBUFFER primitive) to get the required data, instead of multiple ptrace system calls with original ptrace. With the increase in batch size, slowdowns are reduced significantly for all applications. When batch size is increased to 64, the slowdowns of *Apache*, *CVS*, and *PostgreSQL* with improved ptrace are reduced to 1.9%, 0.8%, and 0.5% respectively. For normal applications, such small slowdowns are acceptable.

Related Work

As systems are becoming more complex and problem diagnosis is taking longer and requiring more expertise, quite a few number of related works, that we describe in the *Problem Diagnosis and Resolution* section, have attempted to automate problem diagnosis and resolution. The general approach we have taken to automate problem diagnosis in this work – capturing and utilizing application's runtime behavior – has also been applied to other areas such as debugging and intrusion detection, which we cover in the *Debugging* and *Intrusion Detection* sections, respectively.

Problem Diagnosis and Resolution

A general approach to diagnosing and solving application problems, especially those caused by misconfiguration, is to regularly checkpoint system states and keep track of state changes. For example, Strider [23] takes periodic snapshots of the Windows Registry. When a problem occurs, recently changed or new registry entries are presented as potential root causes. Chronus [25] and FDR [20] also take into account of changes in other system states, not just in the Windows Registry. FDR actually records every event that changes the persistent state of a system. While such system-wide approach is generally fairly comprehensive when it comes to recording changes, filtering out noises (i.e., unrelated changes) and pinpointing the exact root cause can sometimes be difficult. On the other hand, the approach we have taken is very application-specific. We only consider those changes that are known to have an impact on the application that we are diagnosing.

Yuan, et al. [26] is the most closely related work to ours. They try to match the system call sequence of a faulty application with that of a set of known (Top100) problems. When a match is made, the precooked solution to that problem is presented to the user. One problem with this approach is that there is a huge number of different applications, and for each application, there are many possible problems. As a result, the 80-20 rule might not hold true here, which means building a knowledge base of only the Top100 problems might not be sufficient. Additionally, there are a few problems with comparing only system call *sequences*, which we have discussed in the *Application Signatures* section. In our work, we address these problems by converting system call sequences to graph structures. PeerPressure [22] is closely related to the Strider work, also looking at the Windows Registry. It goes a step further and uses statistical methods to compare application-specific Windows Registry entries across many machines to detect abnormal entries. However, this work is limited to only Windows platform and problems caused by mis-configuration in the Windows Registry.

AutoBash [17] is a set of interactive tools to deal with misconfiguration problems. It uses OS-level speculative execution to track causal relationships between user actions and their effect on the application. Fundamentally different from other related works in this section and ours, AutoBash does not monitor historical changes in system and application states in order to find root cause. Instead, it relies on the user to have sufficient expertise in finding the root cause and records the actions taken, in case the same problem occurs again in the future. Users are also required to define predicates specifying what is the correct behavior of an application. These can sometimes be difficult and time consuming to define. In our approach, the correct behavior of an application is already captured by its runtime signatures.

Debugging

Capturing and discovering program runtime invariants are important to programmers when debugging. Various tools [2, 9, 10, 16, 7] are developed for this purpose. Daikon [2] detects invariants based on the values of a set of tracked expression at various program points such as reading or writing a variable, procedure entries and exits. DIDUCE [9] hypothesizes invariants that a program obeys in its execution and gradually relaxes the hypothesis when it observes a violation. These tools usually instrument an application at a very fine granularity to track its "internal" problems. As a result, slowdown can be as much as a hundred times slower or more, which is still acceptable during debugging.

Our tool focuses on diagnosing problems after an application has been released and works while the application is being used. Therefore, low overhead is the key for such tool to be pragmatic, which we have demonstrated in the evaluation of our tool. Furthermore, we do not require having application's source code and monitor the application using a black-box approach. This allows our tool to work also with commercial software which almost always do not have accompanying source code available.

Intrusion Detection

In security area, system calls are commonly traced to detect intrusions [4, 11, 24, 21, 3], where patterns detected in a system call sequence are most important, and other information, such as return value, parameters, and error code, are less so. Intrusion patterns are relatively easier to detect than that of a functional problem of an application, which can happen anywhere in the application and caused by almost anything. Therefore, for problem diagnosis, more detailed information and more types of information are needed to perform accurate diagnosis. And, at the same time, we need to incur as little overhead as possible; like intrusion detection systems, our tool is meant to run alongside of applications. David and Drew build nondeterministic pushdown automata for system calls made by applications [21], which are very similar to system call graphs in our approach. However, they build the automata to have a complete coverage of all the possible execution paths to avoid false alarms. In our approach, we only need to have common execution paths in our signature bank to detect anomalies.

Limitations

Our application diagnosis approach and implementation does have a number of limitations. For example, we do not currently address the problem of how to label a particular application execution trace as faulty. Currently, we rely on a manual indication from the user that invokes the problem diagnosis process. We also adopt a somewhat conservative approach in the amount of information that is collected in application traces. More analysis is needed to identify the minimum set of data that provides a high degree of accuracy for diagnosing common problems. More complete information in signature data is likely to improve the chances that new problems can be diagnosed. Finally, our results, while representative of widely used applications and real problems, are nevertheless limited to a few case studies.

Despite these limitations, we believe that this approach for problem diagnosis represents a promising step toward automating application problem solving, and could lead to significant time (and hence, cost) savings in enterprise IT environments.

Conclusions and Future Work

We have proposed an automatic approach to diagnose application faults. Our approach finds problem root causes by capturing the run-time features of normal application execution in a *signature* and examining the faulty execution against the signature. We have implemented our approach in an user level tool and evaluated it using real application problems that demonstrate that the approach can accurately diagnose most of these problems. We have tested both the space and time overheads of deploying the diagnosis tool, and though the impact on application response time is high, we have proposed and tested a method that significantly reduces it.

Currently our approach builds application signatures on each individual computer system. It is difficult for an user to obtain complete signatures for an application. By exchanging and sharing signatures built on multiple computer systems, users can have more complete signatures to cover more problems. As future work, we plan to explore approaches to share signatures across hosts (e.g., inspired by [22]). When we build signatures for an application on a host, much information specific to that host is included into its signature, such as UID and GID the application is running as, size and last modification time of its configuration files, etc. To share signatures across hosts, some conversion may be required. For example, if the UID has been considered as a piece of signature in a host and we want to share the signature to another host, we have to replace it with the corresponding UID on the target host.

We have evaluated our approach with a number of real problems in a testbed setting, but also plan to evaluate its effectiveness and costs in live deployments, such as campus computer labs.

Author Biographies

Xiaoning Ding is a Ph.D student in the Computer Science and Engineering Department at the Ohio State University. His current interests are in system approaches to improve application performance and maintainability. He can be reached at dingxn@cse.ohio-state.edu .

Hai Huang is a Research Staff Member at IBM T. J. Watson Research Center. His interests are in power management, system and application management, and virtualization. He earned his Ph.D. degree in Computer Science and Engineering from the University of Michigan. He maybe reached at haih@us.ibm.com .

Yaoping Ruan is a research member at IBM T. J. Watson Research Center. His research interests include data center infrastructure management, server application performance analysis and optimization. He can be reached at yaoping.ruan@us.ibm.com .

Anees Shaikh manages the Systems and Network Services group at the IBM T. J. Watson Research Center. His interests are in network and systems management, Internet services, and network measurement. He earned B.S. and M.S. degrees in Electrical Engineering from the University of Virginia, and a Ph.D. in Computer Science and Engineering from the University of Michigan. He may be reached at aashaikh@watson.ibm.com .

Xiaodong Zhang is a professor of computer science and engineering at the Ohio State University. His research interests are in the areas of computer and distributed systems. He may be reached at zhang@cse.ohio-state.edu.

Bibliography

[1] The Apache Group, *The apache HTTP Server Project*, http://httpd.apache.org/ .

[2] Ernst, Michael D., Jake Cockrell, William G. Griswold, and David Notkin, "Dynamically Discovering Likely Program Invariants to Support Program Evolution," *IEEE Transactions on Software Engineering*, Vol. 27, Num. 2, pp. 99-123, 2001.

[3] Feng, Henry Hanping, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong, "Anomaly Detection Using Call Stack Information," *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, p. 62, Washington, DC, USA, 2003.

[4] Forrest, Stephanie, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff, "A Sense of Self for UNIX Processes," *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, p. 120, Washington, DC, USA, 1996.

[5] Massey Jr., Frank J., "The Kolmogorov-Smirnov Test for Goodness of Fit," *Journal of the American Statistical Association*, Vol. 46, Num. 253, pp. 68-78, 1951.

[6] Garbani, Jean-Pierre, Simon Yates, and Sarah Bernhardt, *The Evolution of Infrastructure Management*, Forrester Research, Inc., October, 2005.

[7] Ha, Jungwoo, Christopher J. Rossbach, Jason V. Davis, Indrajit Roy, Hany E. Ramadan, Donald E. Porter, David L. Chen, and Emmett Witchel, "Improved Error Reporting for Software that Uses Black-Box Components," *PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 101-111, New York, NY, USA, 2007.

[8] Haardt, M. and M. Coleman, *ptrace(2)*, 1999.

[9] Hangal, Sudheendra and Monica S. Lam, "Tracking Down Software Bugs Using Automatic Anomaly Detection," *ICSE '02: Proceedings of the 24th*

International Conference on Software Engineering, pp. 291-301, New York, NY, USA, 2002.

[10] Harrold, Mary Jean, Gregg Rothermel, Kent Sayre, Rui Wu, and Liu Yi, "An Empirical Investigation of the Relationship Between Spectra Differences and Regression Faults," *Software Testing, Verification & Reliability*, Vol. 10, Num. 3, pp. 171-194, 2000.

[11] Hofmeyr, Steven A., Stephanie Forrest, and Anil Somayaji, "Intrusion Detection Using Sequences of System Calls," *Journal of Computer Security*, Vol. 6, Num. 3, pp. 151-180, 1998.

[12] Huang, Hai, Raymond Jennings, Yaoping Ruan, Ramendra Sahoo, Sambit Sahu, and Anees Shaikh, "PDA: A Tool for Automated Problem Determination," *Large Installation System Administration Conference (LISA 2007)*, Dallas, TX, December 2007.

[13] LXR, *Linux cross-reference*, http://lxr.linux.no/ .

[14] PostgreSQL Global Development Group, *PostgreSQL: The World's Most Advanced Open Source Database*, http://www.postgresql.org/ .

[15] Price, Derek Robert, *CVS: Concurrent Versions System*, 2006. http://www.nongnu.org/cvs/ .

[16] Renieris, M. and S. Reiss, "Fault Localization with Nearest Neighbor Queries," in *Proceedings of 18th IEEE International Conference on Automated Software Engineering*, pp. 30-39, 2003.

[17] Su, Ya-Yunn, Mona Attariyan, and Jason Flinn, "AutoBash: Improving Configuration Management With Operating System Causality Analysis," *SOSP '07: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, pp. 237-250, New York, NY, USA, 2007.

[18] Transaction Processing Performance Council, *TPC-H*, http://www.tpc.org/tpch/ .

[19] Trent, Gene and Mark Sake, "WebSTONE: The First Generation in HTTP Server Benchmarking," 1995, http://www.mindcraft.com/webstone/paper.html .

[20] Verbowski, Chad, Emre Kiciman, Arunvijay Kumar, Brad Daniels, Shan Lu, Juhan Lee, Yi-Min Wang, and Roussi Roussev, "Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management," *OSDI '06: Proceedings of the Seventh Symposium on Operating Systems Design and Implementation*, pp. 117-130, Berkeley, CA, USA, 2006.

[21] Wagner, David, and Drew Dean, "Intrusion Detection via Static Analysis," *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, p. 156, Washington, DC, USA, 2001.

[22] Wang, Helen J., John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang, "Automatic Misconfiguration Troubleshooting with PeerPressure," *OSDI'04: Proceedings of the Sixth Conference on Symposium on Operating Systems Design & Implementation*, p. 17, Berkeley, CA, USA, 2004.

[23] Wang, Yi-Min, Chad Verbowski, John Dunagan, Yu Chen, Helen J. Wang, Chun Yuan, and Zheng Zhang, "Strider: A Black-Box, State-Based Approach to Change and Configuration Management and Support," *LISA '03: Proceedings of the 17th USENIX conference on System administration*, pp. 159-172, Berkeley, CA, USA, 2003.

[24] Warrender, Christina, Stephanie Forrest, and Barak A. Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *IEEE Symposium on Security and Privacy*, pp. 133-145, 1999.

[25] Whitaker, Andrew, Richard S. Cox, and Steven D. Gribble, "Configuration Debugging as Search: Finding the Needle in the Haystack," *OSDI'04: Proceedings of the Sixth conference on Symposium on Operating Systems Design & Implementation*, p. 6, Berkeley, CA, USA, 2004.

[26] Yuan, Chun, Ni Lao, Ji-Rong Wen, Jiwei Li, Zheng Zhang, Yi-Min Wang, and Wei-Ying Ma, "Automated Known Problem Diagnosis with Event Traces," *EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pp. 375-388, New York, NY, USA, 2006.

Petascale System Management Experiences

Narayan Desai, Rick Bradshaw, Cory Lueninghoener, Andrew Cherry, Susan Coghlan, and William Scullin – Argonne National Laboratory

ABSTRACT

Petascale HPC systems are among the largest systems in the world. Intrepid, one such system, is a 40,000 node, 556 teraflop Blue Gene/P system that has been deployed at Argonne National Laboratory. In this paper, we provide some background about the system and our administration experiences. In particular, due to the scale of the system, we have faced a variety of issues, some surprising to us, that are not common in the commodity world. We discuss our expectations, these issues, and approaches we have used to address them.

Introduction

High-performance computing (HPC) systems are a bellwether for computing systems at large, in multiple regards. HPC users are motivated by the need for absolute performance; this results in two important pushes. HPC users are frequently early adopters of new technologies and techniques. Successful technologies, like Infiniband, prove their value in HPC before gaining wider adoption. Unfortunately, this early adoption alone is not sufficient to achieve the levels of performance required by HPC users; parallelism must also be harnessed.

Over the last 15 years, beowulf clustering has provided amazing accessibility to non-HPC-savvy and even non-technical audiences. During this time, substantial adoption of clustering has occurred in many market segments unrelated to computational science. A simple trend has emerged: the scale and performance of high-end HPC systems are uncommon at first, but become commonplace over the course of 3-5 years. For example, in early 2003, several systems on the Top500 list consisted of either 1024 nodes or 4096-8192 cores. In 2008, such systems are commonplace.

The most recent generation of high-end HPC systems, so called petascale systems, are the culmination of years of research and development in research and academia. Three such systems have been deployed thus far. In addition to the 556 TF Intrepid system at Argonne National Laboratory, a 596 TF Blue Gene/L-based system has been deployed at Lawrence Livermore National Laboratory, and a 504 TF Opteron-based system has been deployed at Texas Advanced Computing Center (TACC). Intrepid is comprised of 40,960 nodes with a total of 163,840 cores. While systems like these are uncommon now, we expect them to become more widespread in the coming years.

The scale of these large systems impose several requirements upon system architecture. The need for scalability is obvious, however, power efficiency and density constraints have become increasingly important in recent years. At the same time, because the size of administrative staff cannot grow linearly with the system size, more efficient system management techniques are needed.

In this paper we will describe our experiences administering Intrepid. Over the last year, we have experienced a number of interesting challenges in this endeavor. Our initial expectation was for scalability to be the dominant system issue. This expectation was not accurate. Several issues expected to have minor impact have played a much greater role in system operations. Debugging, due to the large numbers of components used in scalable system operations, has become a much more difficult endeavor. The system has a sophisticated monitoring system, however, the analysis of this data has been problematic. These issues are not specific to HPC workloads in any way, so we expect them to be of general interest.

This paper consists of three major parts. First, we will provide a detailed overview of several important aspects of Intrepid's hardware and software. In this, we will highlight aspects that have featured prominently in our system management experiences. Next, we will describe our administration experiences in detail. Finally, we will draw some conclusions based on these experiences. In particular, we will discuss the implications for the non-HPC world, system managers, and system software developers.

System Background

The Blue Gene architecture is unusual among computing platforms. It is a completely integrated system, including computational resources, management infrastructure and multiple networks that interconnect them. Much of the hardware and software used in these systems are purpose built. Both of these approaches are at odds with the prevailing trends in commodity clusters. We will provide basic background relating to system management on Blue Gene systems; more detailed information can be found elsewhere [1]. In this section, we will provide an overview of Intrepid's hardware configuration. From there will proceed to a description of relevant system architecture details. Finally, we detail the control system, a nerve center for system management. The control system is the setting for all administration on the compute complex.

Intrepid Configuration

Intrepid is a 40 rack Blue Gene/P system with a pset size of 64. This system is comprised of nearly 160,000 cores, with a peak performance of 556 TF. The compute complex is managed by a single service node. A 10 gigabit data network connects I/O nodes in the compute complex with 136 file servers and several other storage resources. This network is comprised of 512 port switches assembled in a non-blocking configuration to provide 1100 client ports. The system includes 17 Data Direct 9900 storage arrays. Each array is connected to eight fileservers via direct-connected Infiniband; each can provide fail over support for the other seven. This architecture is shown in Figure 1. The service infrastructure is modestly sized, and almost exclusively uses traditional hardware and software.

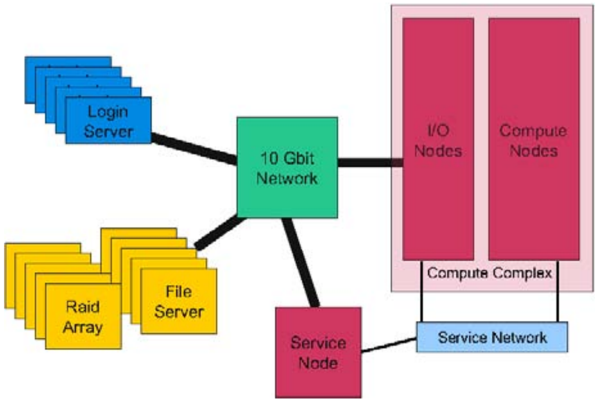


Figure 1: Intrepid system architecture.

Intrepid is a capability HPC system, operated as a part of the US Department of Energy INCITE program [2]. This means that it is intended to run large-scale jobs. Individually, these jobs often consume a large portion of the machine. It is common to see 32,768 core and 65,536 core jobs on Intrepid, occasionally at the same time.

System Architecture

Blue Gene/P compute nodes use quad core PowerPC 450 processors, with access to multiple specialized networks. Compute jobs have access to a mesh network that can be connected into a 3-D torus for collective communication. Also, there is a tree network used to connect compute nodes to dedicated I/O nodes. I/O nodes are connected to fileservers using 10 gigabit ethernet. All compute and I/O nodes also have access to a dedicated management network. This network is used for node booting, diagnostics, and monitoring.

The system is partitioned for each user job. Each partition contains a distinct set of compute and I/O nodes, and the associated mesh network resources. The mesh network must be built into a large rectangular solid. This limits the valid combinations of node resources. Provided sufficient resources are available, a mesh network can be wrapped into a full 3-D torus, providing better network performance. Each of these constraints limit the ways that the system can be allocated, and provide a complicated set of variables when debugging application problems.

The BG/P Control System

The BG/P control system has three main tasks: partition control, job control and system monitoring. It is a single, monolithic process that is run on the system service node. It stores large amounts of data into a DB2 database, run on the same service node. In this section, we will describe each function in turn.

As we mentioned earlier, partitions are the entities where jobs can be run. The architecture of the system depends on a detailed understanding of the topology of the constituents of a partition. This means that partitions are typically rebooted between jobs, upon either user or partition changes. For example, if two jobs were running on a small pools of resources, and these pools needed to be joined for a new job, all of these resources would need to be rebooted before job execution could occur.

The control system plays three main roles in the partition allocation and boot process. It stores and validates the partition configuration, it reserves that hardware used in an active partition, and it implements the partition boot process. The first two of these are straightforward data management issues, while the partition boot process is a little more subtle.

During partition boot, a partition-specific OS image is served to all nodes. This OS image contains all configuration data needed to properly configure the system. The service network internal to each pset implements a hardware broadcast, so the service node need only send the OS images to each node card in the partition; from there it can be broadcast to all nodes. Simultaneously, the control system serves a different set of OS images to the I/O nodes in the partition. When the boot process is complete, each compute node stands ready to execute a user application, and all I/O nodes have mounted all filesystems that can be used by a user job. Also, all I/O nodes run a compute node I/O daemon, *ciod*, that is responsible for executable loading and the proxying of I/O for user jobs.

Once node boot has completed, the user's job executable is loaded, via the ciod. The control system sends the user executable to each I/O node in the partition, which, in turn, loads the user executable on each compute node. Once a job starts on the partition, the ciod begins its main task of proxying I/O requests. When a user process on a compute node performs an

I/O related system call, this call is packaged up and sent to the I/O node, where the ciod performs it and sends the request back to the compute node.

The final responsibility of the control system is to monitor the service network for RAS events. Compute and I/O nodes can directly issue these events to the low-level service network. These events signal a variety of problems, including correctable and uncorrectable hardware errors, and some software errors. The data available from this interface substantially improves the visibility into the compute complex.

Service Infrastructure Management

The service infrastructure is a fairly standard set of systems, both in terms of hardware and software. All of the hosts run a standard version of Suse Enterprise Linux and are managed using a set of standard management tools. Configuration management is nicely handled by Bcfg2 [3], while Nagios [4] is used for monitoring. The service infrastructure is only about 200 systems, and doesn't pose any scalability problems. Most of those 200 or so systems are fileservers, which have a standardized set of software and services.

The service infrastructure is largely run of the mill. This highlights another unique aspect of this system. On typical large scale systems, standard management tools have reach across the whole system. On Intrepid, the combination of specialized and commodity hardware and software make this impossible. Different software and methods must be used on the different parts of the system. This poses a cross-training and coverage problem.

Experiences and Observations

We took delivery of our first BG/L system in January of 2005. This system was comprised of a single rack, with 1024 dual-core nodes. This system was run with a combined set of research and production computation goals and was retired mid-summer 2008. We found this system to be robust and popular with users; these good experiences provided the justification for Intrepid and other associated smaller BG/P systems.

During the summer of 2007, we got initial access to prototype Blue Gene/P systems at IBM. These systems, still under development at the time, were used to port software, both applications and system software, to the new architecture. We also used this opportunity to gain experience with the new control system and runtime environment.

In October of 2007, we took delivery of our first batch of BG/P racks, with delivery and assembly continuing until December. Friendly users have been present on the system since. At the time of this writing, a small development system, Surveyor, is open for general use, while the larger system, Intrepid has eight compute node racks in production and thirty-two more in "early science" mode. These last thirty-two racks are in the process of transitioning to production.

The administrative team of this system is fairly seasoned; many are veterans of other large HPC centers. In anticipation of Intrepid, we each had concerns about different potential system management issues. Many of these concerns focused around standard concerns in large systems: scalability, fault tolerance, and general robustness. To our surprise, several of the issues that we expected to be problematic were handled well by the system, while others proved to be difficult issues to handle. We will discuss several of these in detail throughout the rest of this section.

Control System Issues

As we mentioned earlier, we had expected to encounter scalability issues with the Blue Gene control system. Each BG/P system is managed by a single instance of the control system running on a single service node. Running a system the size of Intrepid with a single control system seemed to be pushing the limits; we had anticipated trouble. On the contrary, control system scalability, thus far, as not been an issue for us. Instead, control system serviceability has been a bigger issue.

Due to the system's reliance on a single instance of the control system, control system restarts are catastrophic; the system must be drained of jobs and no new jobs can be started until the restart has completed. We encountered several bugs in the control system early in the deployment process that caused critical failures, requiring a restart control system to reclaim resources. While these issues have been fixed, we are still vulnerable to this behavior when control system problems occur. When the control system goes south due to unexpected system behavior, the entire system must be idled.

The control system workload has three major components: partition booting, RAS messaging, and state management. Increases in system size have the expected effect on the control system workload; both booting and RAS messaging grow with the system size. Job size and length also have a dramatic impact on the control system workload. The boot process scales non-linearly (in a good way) with the number of nodes, so a single large job boots more quickly than two half-sized jobs. Job length controls how often the boot process and job setup and teardown occur, so longer jobs are gentler on the system.

In this section, we discuss issues related to control system scalability and serviceability. We proceed through the three major areas of the control system, describing our workload, decisions we have made and their combined impact.

Node Boot

One of the main tasks performed by the BG/P control system is providing the node boot infrastructure. The boot process on BG/P systems is highly parallelized, and hence scalable. However, this scalability comes at a cost. The boot process consists of a variety of tasks operating in parallel; any of these can fail. The robustness of the boot process has been improved through the use of simple mechanisms for system boot; due to the integrated nature of this system, standards like PXE need not be applied. Instead, the control system uses a Joint Test Action Group (JTAG) network to load kernels on the compute complex. This network provides low-level access capabilities to individual hardware components in the compute complex. It has a limited hardware broadcast capability which greatly improves the scalability of the boot process.

After the kernels have been loaded onto the compute complex, the compute nodes boot a custom lightweight kernel, called the CNK (compute node kernel). The I/O nodes boot Linux, start services consumed by compute nodes, bring up external networking, and mount all filesystems. A majority of boot problems occur when the compute complex is interfaced with the external service infrastructure. In particular, these problems happen during network and filesystem bringup.

As we mentioned earlier, due to limitations in the BG/P runtime system, nodes must be rebooted between jobs if their partition configuration changes. In light of this restriction, and other factors discussed later, we have opted to reboot nodes between each job. Due to this operational decision, partitions are rebooted in advance of each job. The boot process is sufficiently quick to make this convenient; a 16 rack (64K task) partition boots in less than 5 minutes. This performance is a marked departure for the boot process on commodity systems, where network boots are considerably slower and more fragile.

Despite this relatively low cost, this approach has other tradeoffs. A higher load is placed on the control system, due to the increased frequency of node boots. Also, while the boot process itself is scalable, I/O and network related problems do occasionally occur during the boot process. An increased frequency of node boots increases the possibility of this occurrence.

On the plus side, per-job node reboots reduce the ability of latent state to impact jobs; the system starts from a clean slate for each job. This makes job performance consistent and reproducible. Moreover, it minimized the occurrences of idiosyncratic software problems. On the balance, this decision has been the right one; we are convinced this policy has allowed us to avoid far more problems than it caused.

Debugging boot problems requires a wide view of the activities of all of the components included in a partition. Failures fall into two basic categories: single component failures and workload dependent failures. Single component failures, of course, are far easier to diagnose that workload dependent failures; however, even single component failures can be difficult to locate.

The most frequent sort of single component failures cause network or filesystem failures on I/O nodes. I/O nodes use a simplified Linux boot process that runs a series of configuration scripts. The boot process is fundamentally serial; I/O nodes independently start up, and are not made aware of failures on other nodes. Even if failures can be locally detected, they are not adequately communicated to other components in the system. Direct access to the RAS system is not conveniently available from this context, however, enhancements in this area are forthcoming.

Load-related errors can also cause issues on the network during large-scale partition boots. Because I/O nodes are rebooted along with compute nodes for each job, the network must be able to properly cope with large numbers of nodes frequently leaving and joining the network. All the while, I/O is being performed for other active jobs. Workloads of this sort have triggered a variety of switch firmware bugs, all of which have been difficult to replicate. When these problems occur, they are painful to troubleshoot and resolve. We currently have no mechanisms that help us with this process.

RAS Messaging

The BG/P control system includes a scalable logging framework for RAS events. These events are generated in a variety of conditions corresponding to hardware and software failures. When software failures occur, they frequently occur across an entire job. Because large jobs are common on Intrepid, these errors can cause up to 160,000 RAS messages per incident. This count corresponds to a RAS event per task on a full-system job. Even a more modest job size could easily result in upwards of 32,768 events for a single error. The control system also stores a variety of data about system hardware and current and historical jobs.

This volume of logging data cannot be retained indefinitely. Even though Intrepid has been in operation for less than a year, we already face difficult data retention policy questions. The amount of stored data has a direct impact on the performance of the control system; as more data is added, queries take longer. Ideally, data would be retained forever, to allow for long-term trend analysis; this is clearly not possible.

The introduction of a data-intensive system like this into system management is an abrupt departure from the usual approach used on HPC systems. We were frankly unprepared to deal with data and transaction volumes at this level. In the last few months, we have added a dedicated DBA to the system management staff; this appears to be improving the situation.

The scalability of the RAS messaging infrastructure itself has not been a problem; during acceptance tests we were able to successfully simulate intense storms of RAS messages without issue. The system is even responsive during these situations.

The RAS subsystem is a critical resource for debugging system issues, though the volumes of data involved make this task somewhat difficult. These issues are discussed in detail in Section RAS-based Debugging.

Partition State Management

The final role of the BG/P control system is to track compute system states through the partition and job lifecycles. This process is scalable, however, it is the largest current source of serviceability problems. In some cases, partitions end up in a state where they cannot be reclaimed. When this occurs, only a full control system restart can restore the partition to operation. In order to perform a control system restart, the machine needs to be drained of all jobs, resulting in diminished utilization. This problem has posed the largest issue of all of the control system issues mentioned in this section. While IBM has been quite responsive, and bugs have usually been fixed relatively quickly, the potential for trouble remains. This is the main detriment caused by the control system's implementation as a single scalable component.

Fault Management

Large scale systems are more sensitive to hardware failure than decoupled systems. This sensitivity occurs due to the properties of parallel workloads; many HPC applications will fail outright upon single component failure. Moreover, the increase in component counts in large-scale systems cause failures to occur more often.

Some of the fault management work on Intrepid consists of traditional component diagnosis and replacement, however, the scale and workload of the system makes failure isolation difficult. Because of this, more sophisticated techniques are to discover failing components.

Intrepid also has a number of useful features for fault management. The compute complex ships with comprehensive hardware diagnostics. The RAS system provides detailed information about unexpected hardware and software events at runtime.

In this section, we will discuss role of system diagnostics in system management operations, the processes employed between runs of the diagnostic system and use of the RAS system in debugging.

System Diagnostics

The Blue Gene system ships with a set of diagnostic routines. These tests verify the proper function of all components in the system. While the full set of tests are quite comprehensive and accurate, they are quite time consuming to run. A single rack full diagnostics takes on the order of an hour to run diagnostics and currently a maximum of two racks may be run in parallel. The cost to run regular full diagnostics on a system this size is simply unreasonable at this point. In order to guide regular diagnostics, IBM has developed a smaller, general health check that can be run in parallel across the full 40 rack system in a reasonable amount of time (under two hours). This general test can call out problem areas which can later have a full diagnostics test suite run against them.

These diagnostics greatly improve our lives; at the same time, they present us with a difficult choice. Diagnostics are able to discover marginal hardware before it has failed outright, so we benefit from frequent diagnostics. On the other hand, diagnostics monopolize system resources, reducing the number of cycles delivered to users. This issue is similar to the one faced by users choosing a checkpoint frequency in parallel jobs [5].

The addition of the general health check capability and more accurate assessments of hardware failure rates will allow us to pick a reasonable frequency for system diagnostics. That said, hardware failures will always occur in the intervals between diagnostic runs, so manual troubleshooting techniques are also needed.

Diagnostic Search

Regardless of the quality and frequency of system diagnostics, unexpected failures will still occur. When they do, users report erroneous system behavior. This behavior must be diagnosed by the system administrators. Parallel systems and workloads are problematic in this regard. Large numbers of components operate in parallel to accomplish a given task; a single failure in this context frequently causes overall process failure.

Much of the BG/P's scalability results from aggregation. While aggregation is good for scalability, it can have a detrimental effect on diagnostic procedures. Grouping individual components into parallel, aggregated process often obscures the source of failures. Incorrect collective behavior is observed, but that alone does not provide enough data to isolate the cause.

In this case, we use a binary search of system hardware to isolate the failing component. Luckily, most tasks can be run on partitions of arbitrary size, so we can use the same test on smaller partitions until a cause emerges. This technique is quite effective; it provides a fast path to isolate a faulty midplane. Once a midplane is isolated, system diagnostics can be performed without affecting other midplanes. This combination of search processes and system diagnostics has been an effective tool for faulty hardware isolation, even when problems occur in large groups of components.

RAS-based Debugging

Problem identification is substantially harder on Intrepid due to the system architecture and workload. Many operational issues on Intrepid do not result in a single log message that describes the issue. Rather, problem identification consists of the correlation of log data from a variety of locations. In addition to RAS events and control system data, many other components in the system also log information in a variety of formats. I/O nodes and service infrastructure nodes run Linux and produce standard logs.

Identification of problems in real-time or even near-real-time requires the correlation of events from a large number of sources potentially entering the system at a high rate. While several correlation frameworks exist and are publicly available [6, 7], correlation at this scale is an open research issue. Our current approaches to these problems use small volumes of data, heuristics, and intuition, to a reasonably good effect. However, progress on this front would greatly improve our ability to recognize subtle problems in a more timely manner.

Management Effort

Intrepid is administered by a team of three full-time system administrators and a DBA. Assistance from others provides for off-hours coverage and user support. Two of these administrators are primarily concerned with the compute complex and control system, while the other manages the service infrastructure. This division is purely practical; the service infrastructure requires much more daily administration effort, scaled for system size, than the compute complex does.

Considering the relative sizes of the systems, this ratio of administration effort is surprising; the 40960 node compute complex only takes twice the operation effort as a 200 node commodity system. We have determined a variety of factors that play a part in the decreased administration costs of the compute complex. In this section, we will contrast each of these with the traditional model used in commodity systems.

Persistence

One major difference between the service infrastructure and the compute complex is the workload. The compute complex is a consumer of services, and is frequently rebooted. On the other hand, the service infrastructure provides all services consumed by the compute complex. In particular, the compute complex interacts with the control system and several file systems served from storage nodes.

Frequent reboots of the compute complex, as new jobs are run, minimize the amount of problematic state that can be accumulated. This approach is not possible on the service infrastructure as the services provided must persist beyond a single job. This need for continuity makes the service infrastructure far more difficult to administer than the compute complex.

Configuration Complexity

Each half of the system uses a different configuration methodology. When a partition is booted, the same OS image is sent to all compute nodes; a (different) single OS image is sent to all I/O nodes as well. Each of these OS images contains the union of all configuration data needed for all nodes, yet still remains quite small. Frequently these images are less than 16 MB.

By contrast, commodity systems running standard distributions of Linux have a higher configuration requirement. Bcfg2 [3] configurations for Linux systems configured in the service infrastructure contain information about nearly 800 aspects of system configuration; these configuration specifications can be upwards of a megabyte in total. This difference is quite striking; the configuration burden on compute nodes is substantially lower than that of commodity systems.

The reduced complexity of compute complex configurations has both positive and negative repercussions. The compute complex is substantially less agile than we, as administrators, have become accustomed to with commodity systems. In an HPC workload, with relatively small numbers of large jobs, this shortcoming has not yet posed a serious problem to us. We anticipate this to become more of an issue as sites start to use BG/P-style systems for varied workloads.

Moreover, for the reasons described above, all configuration management must be performed incrementally on the service infrastructure, due to service continuity requirements. Incremental reconfiguration processes are clearly more error-prone and resource intensive. Incremental approaches have a much higher burden in terms of compliance monitoring. Due to the frequency of partition reboots in the compute complex, incremental approaches are not needed as drastically.

Hardware Robustness

The compute complex hardware is substantially more robust than even server-grade commodity hardware. The addition of robust diagnostics make the operation of the compute complex considerably easier than the service infrastructure, or any other commodity system. While we face issues in isolating failing hardware in the compute complex, this process still takes less time than maintenance activities in the service infrastructure. As purely anecdotal evidence, we have replaced as many compute nodes as we have serviced fileservers during system operations. This demonstrates the drastic difference between the failure rates in the different parts of the system.

Conclusions

In this paper, we have provided a discussion of our experiences managing Intrepid, a large-scale Blue Gene/P system. Our main operational issues focus around serviceability and fault management. One striking finding was the relative level of effort required to operate the halves of the system. Due to the capabilities provided by the BG/P control system and RAS infrastructure system management efficiency is drastically higher in the compute complex compared with the service infrastructure. For this reason, we anticipate that design choices similar to those made by the BG/P design team will become common in other systems as well. System administrators will have no choice but to cope.

Applicability to Other Environments

System sizes have been on the rise for the last two decades and this trend shows no sign of slowing. Meanwhile, the relative costs of administration and maintenance of these systems has continued to grow.

The relatively low operations cost and high scalability of integrated systems like the IBM Blue Gene/P or other integrated MPP systems make them appealing for wider scale use. However, these benefits come at a substantial complexity cost. Nonetheless, IBM has begun to adapt Blue Gene/P systems to non-HPC workloads [8].

Virtualization and cloud computing have become popular recently. Large scale systems, whether composed of real hardware or virtual nodes pose many similar problems in administration. These systems, once they grow to a sufficient size, will experience many of the same management and problem determinations issues that we have seen on Intrepid.

Consolidated systems such as these will introduce complex interdependencies between components and even between nodes in some cases. I/O starvation has long been a problem in the HPC space. It has become a pressing issue in virtualized environments as well. As the number of entities competing for resources in virtualized environments grows, administrators must be able to diagnose multi-system, workload-dependent issues in an effective way.

Moreover, administrators are frequently tasked with the operation of systems of increasing size without corresponding increases in staffing. These issues will be exacerbated in virtual environments, where new virtual machine instances are effectively free.

Implications for System Software and Tools

Our experiences have a variety of implications for system management tool developers. Scalability has long been the the boogy-man of the system software world. Perhaps it isn't as frightening as everyone has been assuming. Certainly there are cases where scalability is a dire concern, but this is not universally the case. A more nuanced understanding of service scalability is needed.

At the same time, in some cases, scalable operations are required. These mechanisms cost. For example, the boot process uses a hardware broadcast capability in order to scale. In terms of configuration, this approach is a step backwards to the days of shared NFS root file systems. In this case, we have lost a substantial amount of flexibility, compared to modern Linux systems.

Data analysis is much more difficult than we had initially anticipated. Realtime analysis of our volume of data is not possible with the current generation of publicly available analysis tools such as Sec [6]. We are unsure that commercial tools will scale to this level, although large scale web service providers probably have home-grown tools in this space. This is one area where a scalable, most likely parallel, approach will be needed.

Collective approaches to system management need to be developed. While some initial work in this area has occurred [9, 10], collective approaches to system management have not yet become convenient enough to deploy in practice. For example, a reasonable distributed control mechanism could be used to provide fine-grained configuration management capabilities in a scalable fashion. Similarly, distributed approaches to data correlation could provide the scaling needed on systems of this sort.

Finally, much of the configuration complexity present in traditional systems have been eliminated outright from this architecture. In some cases, we find functionality missing, however, on balance, we have found the system quite usable. Might we be near the end of our collective configuration nightmare?

Author Biographies

Rick Bradshaw is currently a Senior Systems Administrator for the Mathematics and Computer Science Division of Argonne National Laboratory. His interests are mainly focused around configuration management, and experimental and HPC computing. He holds a bachelors of Computer Science from Edinboro University of Pennsylvania, and is currently working on a Masters in Computer Science at the University of Chicago.

Andrew Cherry is a UNIX system administrator who has worked on a variety of systems, in both commercial and non-commercial environments. He is currently an HPC system administrator at Argonne National Laboratory, where he is responsible for day-today management of Argonne's Blue Gene/P supercomputer.

Susan Coghlan has worked on parallel and distributed computers for 20 years, from developing scientific applications, such as her work on a model of the human brain at the Center for NonLinear Science at Los Alamos to managing ultra-scale supercomputers like ASCI Blue Mountain, a 6144 processor supercomputer at Los Alamos National Laboratory. In her current role as Associate Division Director and Director of Operations for the Argonne Leadership Computing Facility, she is responsible for the installation and operation of the world's fastest open science computer (Top500, June 2008) the ALCF's 557TF Blue Gene/P production system. She is well known within the HPC community, and has presented numerous tutorials, lectures, and papers on her work. When not fiddling with some of the world's largest computers, she does so with other Irish traditional musicians.

Narayan Desai is a researcher in the MCS Division of Argonne National Lab. He specializes in system software, particularly as it relates to system management, fault tolerance and scheduling.

Cory Lueninghoener is an HPC System Administrator with the Leadership Computing Facility at Argonne National Laboratory. When not keeping a 40-rack BlueGene/P system running, he spends time

hacking at tools like Bcfg2. Cory can be reached at lueningh@alcf.anl.gov .

William Scullin is a HPC systems administrator at Argonne National Lab's Leadership Computing Facility. His favorite systems administration tools after Bcfg2 and BlueGene Navigator are macromancy and python, though that may be repetitive. Outside of work, he seeks to raise awareness of tyrophagia.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

This work for the Argonne Leadership Computing Facility was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Bibliography

[1] Gara, A., M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L System Architecture," *IBM Journal of Research and Development*, Vol. 49, Num. 2-3, pp. 195-212, 2005.

[2] US Department of Energy, *Innovative and Novel Computational Impact on Theory and Experiment (Incite) Program*, http://www.er.doe.gov/ascr/incite/index.html .

[3] Desai, N., *Bcfg2 web site*, Argonne National Laboratory, http://trac.mcs.anl.gov/projects/bcfg2 .

[4] *Nagios web site*, Nagios Enterprises, LLC, http://www.nagios.org .

[5] Oliner, A. J., R. K. Sahoo, J. E. Moreira, and M. S. Gupta, "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems," in *IPDPS*. IEEE Computer Society, 2005.

[6] Rouillard, J. P., "Real-time Log File Analysis Using the Simple Event Correlator (Sec)," *LISA*. USENIX, pp. 133-150, 2004.

[7] *Splunk web site*, Splunk, Inc. http://www.splunk.com .

[8] Appavoo, J., V. Uhlig, and A. Waterland, "Project Kittyhawk: Building a Global-Scale Computer: Blue Gene/P as a Generic Computing Platform," *SIGOPS Operating System Review*, Vol. 42, Num. 1, pp. 77-84, 2008.

[9] McEniry, C., "Moobi: A Thin Server Management System Using Bittorrent," *LISA*, USENIX, pp. 253-260, 2007.

[10] Desai, N., R. Bradshaw, A. Lusk, and E. Lusk, "MPI Cluster System Software," *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Springer Lecture Notes in Computer Science, D. Kranzlmuller, P. Kacsuk, and J. Dongarra, Eds., Num. 3241, Springer, pp. 277-286, 2004.

Rapid Parallel Systems Deployment: Techniques for Overnight Clustering

Donna Cumberland, Randy Herban, Rick Irvine,
Michael Shuey, and Mathieu Luisier – Purdue University

ABSTRACT

Automated system deployment frameworks and configuration management systems have been in wide use for a number of years. However, due to increasing pressures to maintain high availability, coupled with the price effects of commodity servers, administrators may be required to deploy large numbers of systems in shorter time frames than is normally possible with available staff. In this paper, we describe a straightforward procedure using commonly-available infrastructure to enable rapid simultaneous deployment of hundreds of machines by temporary staff. As an example of the efficacy of this approach, we present a case study in rapid systems deployment at Purdue University. On May 5th, we deployed Purdue's "Steele" cluster, installing over 500 compute nodes in a single business day.

Introduction

Changing system requirements, additional projects, and life-cycle system replacements all result in deployment of new machines in the data center. Given the current popularity of grid computing and software-as-a-service, and the proliferation of virtual servers within the enterprise, the number of new systems will likely continue to rise.

Modern services, particularly high performance computing centers, often deploy clusters of hundreds, if not thousands, of individual machines. While several custom tool sets exist to aid this sort of environment [1, 2], these tools are usually heavily adapted to their task and may not readily integrate into existing configuration management tools. Also, these tools are often built around a particular market segment, such as the high performance cluster community. As such, they may not be well suited to other uses (e.g., dedicated application servers, web farms, digital rendering clusters, etc.)

Several configuration management tools exist to allow few administrators to manage large collections of machines [3, 4, 5]. These can easily be used to encompass both large-scale system deployments (clusters, render farms, etc.), but do not encompass the actual operating system installation. Other tools must be employed to actually deploy the system, and load the configuration management software.

Most modern operating systems provide some means for repeatable, automatic software installation [6, 7]. These tools can, in some cases, be directed to prepare configuration management software on the new host. However, they assume some external method is used to identify the system being installed – either via a pre-configured service, like DHCP, or by manual network name and address assignment. These are generally quite labor intensive.

By combining the available component tools, we describe a server deployment approach more flexible than current purpose-built tools. With the addition of IPVS [8], an IP-level load balancing service, such an approach is readily able to handle the peak load of hundreds, if not thousands, of new servers being deployed simultaneously. This capability greatly reduces software deployment time, and (with the addition of temporary labor to assist with physical tasks) can enable the installation of large systems in surprisingly little time.

For the remainder of this paper, we present an insight into our motivation behind taking this deployment approach, followed by a more detailed discussion of the difficulties involved in a highly parallel deployment and our implementation to address these. As evidence of the success of this approach, we present a case study in rapid cluster deployment – the recent installation of Purdue's "Steele" cluster. We conclude with the lessons learned from this deployment exercise, and some indications of possible scaling limitations moving forward.

Motivation

Purdue University is home to several active scientific research communities, many of which make use of high-performance compute clusters maintained in central campus data centers. As with many institutions, data center space is at a premium. New systems cannot be deployed without first removing older equipment. In most cases, the out-bound older equipment is still serving a research group the day it leaves. These users are effectively left without service until a new system can be deployed.

In the spring of 2008, Purdue faculty, staff, and administrators began the design and purchase of a 806-node compute cluster (more fully described below).

Data center facilities limitations forced the removal of three existing clusters, with a combined total of over 900 compute nodes. Equipment removal alone would take approximately one week, and required all available staff with familiarity with the facility. Only a small portion of the new cluster could be assembled before the existing equipment's removal.

To reduce the overall impact on Purdue's research community, we developed the techniques described to deploy the majority of the nodes in a single day. The success of this endeavor is covered in more detail in the case study section.

Infrastructure Components

Rapid server deployment depends on automating the installation process as much as possible. Any amount of variation between servers requires manual intervention, and this must be minimized to maximize system administrators' productivity.

The server deployment process can be seen as a three-phase process: Installation of the base system software, identification of the machine (including assignment of network addresses), and merging the new machine into a full configuration management system.

Base System Software

Modern gigabit networks can deliver data in excess of 100 MB/sec, rivaling hard disks and exceeding the speed of any other installation media. To improve deployment times and eliminate administrator time spent swapping media, using a network-based install method is essential.

As mentioned above, most modern operating systems have some sort of network-based installation system. With firmware-level network boot services (such as PXE [9]) on most new hardware, using this is fairly straightforward. Multiple machines can be simultaneously installed from a dedicated PXE configuration, to invoke systems like RedHat's KickStart or a Solaris JumpStart as appropriate.[1] For the remainder of this discussion we will focus on extensions to RedHat's KickStart system, which is in wide use in our environment.

The PXE environment does have one large drawback: in order to uniquely identify each machine being deployed, an administrator must either populate a DHCP server with the MAC address of each machine in question (along with each machine's proper IP address), or separate installation configuration files for each system. While scriptable, selecting the proper configuration for each host is often extremely time-consuming. For this reason, we have chosen to install all deployed systems with a common operating system load at first boot, using temporary network addresses. Network address assignment, and proper machine identification, can be done as a second pass through the mostly-installed systems (as described in the Machine Identification section).

[1]The PXE boot system can easily be extended to provide for more than a mass-deployment mechanism. In our environment, individual servers being installed are brought to a PXE menu where administrators can select from a number of installation and testing options including memtest, Knoppix [10], and dedicated entries for each operating system and version we have on site.

KickStart Configuration

RedHat's KickStart system is a means to automate the installation and initial configuration of RedHat's Enterprise Linux operating system. Several types of media (e.g., CD-ROM, HTTP, NFS) are supported to deliver system software. In all cases, the install process follows directives listed in a file, `ks.cfg`. The `ks.cfg` file lists key system configuration values (such as the initial root password, timezone configuration, etc.), any network parameters needed, the software packages to be installed from the selected media, and (optionally) a post-install script to execute before rebooting from the OS installer.

In our environment, all server deployment takes place over the network. Our PXE boot environment will cause RedHat's installer kernel (and initial filesystem) to be loaded via TFTP. Linux command-line options, as well as configuration directives in the `ks.cfg` configuration file, are used to ensure that all systems use a random DHCP-provided address during the install process. This lets us use a single configuration to serve hundreds of clients with no customization. At the end of the OS install process, systems are moved to a fixed, production IP address (to correspond with their location in an equipment rack). This process is explained in detail in the machine identification section.

All system software is provided by a local HTTP server, and all RedHat packages to be installed on a server are listed in the `ks.cfg` file. Since HTTP is a unicast service, this does pose a bottleneck for bulk system deployments. However, as this data is only read (never written) during the installation process, it can easily be replicated. Multiple HTTP servers can be clustered to increase the overall bandwidth, provided they are aggregated behind a single network name (to simplify configuration and reduce the amount of customization in `ks.cfg`). Our preferred choice for this is an IPVS-based load balancing cluster (described further in the IPVS discussion, though in practice even a simple DNS record pointing to multiple addresses would suffice.

Once RedHat's KickStart system has loaded and configured the base operating system, the post-install script contained in the `ks.cfg` file is executed. We use this script to properly identify the machine and set its final network address, as described below, and install critical pieces of our cfengine configuration (as described in the configuration management section.

Machine Identification

Without expending a great deal of effort to carefully catalog, place, and install each machine in a large group, a machine's final network address is often not apparent until it has been physically installed. A compute node's identity is often based on its location in an equipment rack – a property that is very difficult to discern from a boxed machine on the back of a truck.

Some systems (e.g., IBM's SP-2 [11]) are able to discern their identity by automatically detecting their location in a specially-wired management network. With additional scripting, this can be translated into node names, network addresses, and useful configuration data. However, this method is usually not an option when using today's low-priced commodity hardware.

Other systems (e.g., OSCAR [1]) use a mechanism to collect DHCP requests as machines are booted, and thus identify systems in order. These systems require machines to be booted serially, though, and prevent increasing the deployment parallelism to the level we would prefer. Instead, we have devised a simple mechanism to touch each machine in sequence, to have each system confirm its proper network address.

In our environment, we extended the kickstart configuration file to include an infinite sleep-wait-probe cycle in the postinstall section before configuring their production IP address. This allows us to start the installation process on many nodes simultaneously without regard to their final naming. The loop will only break if the machine sees the insertion of a USB thumbdrive, as shown in Listing 1, a kickstart script excerpt.

With this in place, an administrator can assign network addresses to a series of machines by simply making a pass through the entire group, inserting a USB thumbdrive into each machine. The thumbdrive itself is not important – we are using the act of inserting this USB device as a means of physically identifying one machine out of hundreds of identical systems. Other simple methods of quick, manual identification (e.g., a key press, CD insertion, etc.) would be equivalent.

Once a machine has been physically identified, it contacts the web server and downloads its proper network configuration. Obviously, the CGI program get_ip.rh.cgi must return a valid network configuration file, then increment the IP to the next production IP address in sequence.

Configuration Management

Many configuration management systems are in widespread use. Assuming that all configuration information (beyond basic system parameters, such as partition layout and operating system version) has been expressed through a site's preferred configuration system, the remainder of a system's deployment should be a very simple exercise.

Purdue's high-performance computing environment uses cfengine [12] heavily. The final step in our customized kickstart configuration is to download a custom shell script to initialize cfengine, and run it at first boot. The script includes an in-line shar archive containing the necessary binaries (and key configuration files) for that architecture. This script is placed at the end of the boot sequence for the new host.

When the newly deployed system first boots, the script will run cfengine twice (first with special options to mitigate site-specific dependencies and ordering problems, then with default options). This configures the system, installs any missing site-specific software, ensures various cfengine-related processes are started at boot time, and applies any outstanding security errata not a part of the initial OS install. Finally, the script removes itself from the boot process and reboots the host. Once rebooted, the system is a production-ready server.

Limitations/Acceleration

IPVS

We felt the throughput capacity of our file serving infrastructure for both kickstart and cfengine initialization needed to be upgraded for massively parallel deployments. Our existing machine for these functions

```
%post
cd /tmp
/sbin/modprobe usb-storage
while (true); do
   if [ dmesg | tail -n 3 | grep 'USB Mass Storage device found' ]; then
      wget http://install-server/cgi-bin/get_ip.rh.cgi
      cp /tmp/get_ip.rh.cgi /etc/sysconfig/network-scripts/ifcfg-eth0
      #
      # configure any other network-related bits here
      #
      ifdown eth0
      ifup eth0
      break
   fi
   sleep 3
   echo "Still waiting for USB thumbdrive..."
done
```

Listing 1: Thumbdrive detection.

was an older dual-processor system, and we were concerned that disk I/O limitations of its aging RAID array would be exceeded during the installation.

By utilizing a load balancing cluster with a round robin distribution algorithm, we segmented the demand down to equal parts for each real server. By placing the cluster behind a single IP address, we avoided having to change any other part of our automated installation infrastructure. We chose to use Linux's IPVS kernel module [8] and a set of freely available tools [13] to create this load balanced cluster and to give it high availability as well.

Purdue's configuration required additional machines for this purpose, so we re-purposed a few existing LDAP authentication servers. As such, in addition to the kickstart and cfengine file sharing responsibilities, the IPVS cluster also now handles LDAP authentication. We ended up with two cluster front ends and four real servers, all newer dual-processor dual-core servers. This gave us theoretically 4 gigabits per second of file transfer rates for the cluster node installations and also gave each node in the cluster only a quarter of the load.

We could have simply used a single larger machine. However, by leveraging the existing LDAP authentication hardware (which was not being overly taxed) and using open source software, our solution required no monetary investment. Since all IPVS-related configuration changes were merged into cfengine while building out the initial server setup, the IPVS cluster itself was rapidly deployed. We simply treated it like any other cluster – PXE boot a new system, automatically install the OS and run cfengine, then manually add it to the IPVS cluster routing table on the front ends.

In addition, if we had determined during a mass deployment that the load was too high for the IPVS cluster, we had the option of utilizing several of the new cluster machines being deployed as additional kickstart/cfengine file transfer nodes. We could have simply installed them as generic servers, labeled them as IPVS real servers in cfengine, and set them to work. The initial cfengine on an IPVS real server does take a little while due to the large repository of install data that must be copied over, but it still was quick enough to have been an option during the install day exercise.

This solution to our increased file transfer needs was cheap, easy, and a very reliable means to scale out our deployment bandwidth.

Squid Cache and RH Proxy

Our compute cluster nodes generally use Redhat's Enterprise Linux [22]. Purdue has a university-wide Redhat Network [23] proxy server that handles registering new OS installs and caches updated software downloads. The existing server was an older dual-processor system, with relatively slow disk array (as compared to current technology). We've seen this system become overloaded when any Purdue IT group would attempt to update more than a couple dozen machines at once.

For the installation day, we felt this system needed to be upgraded as well. We installed a new dual-processor, dual-core server with twice the memory of the old machine, from four to eight gigabytes, in order to handle the heavier I/O traffic. Linux makes aggressive use of unallocated memory to cache storage I/O, so we were hoping this system would yield significant performance improvements.

We briefly looked at putting the squid [14] web cache portion of the RHN proxy server behind the IPVS cluster, but it appeared that the squid server needed to be local to the registered RHN proxy server. In additional, the actual RHN proxy software was licensed for a single machine. As such could not legally be put on each real server, it had to run on a single machine. After testing, we determined that additional proxy servers, or further work integrating the RHN proxy server with our IPVS cluster, would not be necessary for the size of deployments we typically encounter. However, this may be a limiting factor for much larger installations (with thousands of machines).

The larger proxy machine was more than capable of handling the load generated by installing the sizable "Steele" cluster at Purdue. Had it not been able to, we would have simply not run so many initial operating system installs in parallel. We wanted to have every node installed and running by the end of the day, but we were realistic in expecting the possibility of some machines finishing their installs and cfengine configurations overnight. As it turns out, the proxy server never came close to heavy load. We estimate that the system could have easily handled two to three times more load.

Subdivide Networks, Conquer DHCP

One shortcoming of this method is the serial nature of using the USB thumbdrive to number machines. It can quickly become the bottleneck and increasing the number of machines installed via this method only worsens the problem. One method to parallelize is to divide the machines up onto multiple network subnets and give them separate DHCP/PXE entries. Each entry can then use a separate source to obtain the IP address, and can have its addresses assigned independently. During previous large system deployments, we have spanned three subnets and have been able to identify three machines at once. We see no reason why this could not be trivially increased for larger installs, allowing for wider deployment parallelism.

TFTP and PXE Performance

After the infrastructure improvements described above, TFTP remains a single point of contention. Mass machine deployment efforts may direct hundreds of clients to a single TFTP server as each machine loads its operating system installer. In practice, however, this service sees insignificant load – even during our large deployments.

The operating system installer we use, provided with RedHat Enterprise Linux, consists of 6.4 MB of software. On a gigabit ethernet network, that amount of data takes less than a second to transfer – less time than it takes to physically turn on a new machine. If several dozen systems were to start the PXE process at the exact same second, it is conceivable that we might have some contention at the TFTP server's network port. For our purposes, though, that was deemed an acceptable risk.

Case Study: Purdue's "Steele" Cluster

Much of our work to streamline, parallelize, and accelerate system deployment came about as a result of the diminishing quantity of available data center capacity, coupled with the rising pressure to effectively use the remaining space as quickly as possible (as mentioned above). In the spring of 2008, we saw an opportunity to push the infrastructural improvements we describe to the limit of our facilities' capacity.

As mentioned above, Purdue's "Steele" cluster could not be supported in our facilities without the removal of a large number of existing production computing systems. To better accommodate our research community, we opted to deploy as many systems as possible ahead of time, leaving the remainder to be installed in one large batch on May 5th, 2008. To provide more labor during the install day, we recruited volunteers from positions throughout the university, none of whom were familiar with the deployment process. The details behind this installation day, and its successful reception by some of our researchers, are described in the following sections.

Machine Description

The "Steele" cluster consists of 806 dual-processor, quad-core compute nodes, in four different memory and network configurations. Configurations break down as follows:

- 24 nodes, 32 GB memory and both gigabit ethernet and Infiniband network connections
- 41 nodes, 32 GB memory and only gigabit ethernet networking
- 180 nodes, 16 GB memory and both gigabit ethernet and Infiniband network connections
- 561 nodes, 16 GB memory and only gigabit ethernet networking

The three smaller configurations are housed in one room of Purdue's research datacenter, while the largest configuration (with 561 nodes total) is housed in a separate room.

The configuration is connected with a single large ethernet switch, with over 600 available ports of gigabit ethernet. Other configurations are connected to smaller gigabit ethernet switches, which then link to the larger switch via 10 gigabit ethernet connections.

Pre-Deployment Steps

To lessen the service impact to University researchers, we opted to install as many compute systems as possible leading up to the May 5th install day. Unfortunately, due to facilities limitations, this amounted to about 12% of the system's total capacity. In addition, we were able to unbox, rack, and install a number of systems (roughly another 12% of the compute nodes) prior to the general installation – though these systems were not fully deployed, and in many cases had little or no power or network wiring.

In the week prior to the main install day, large sections of our facilities were cleared. Network components were installed and configured, equipment racks and electrical connections were set in place, and ethernet wiring was run from most switch gear to patch panels in each equipment rack.

The Human Factor

Simply put, there were too many systems for our usual systems administration team to handle in a single business day. Without additional help, our lack of manpower alone would force us to prolong this deployment over at least a week, possibly two. However, with the deployment procedure fully automated, we were able to solicit assistance from the rest of the University.

In addition to the obvious need for aid in wiring systems and initiating PXE-based OS installs, we also sought volunteers for a number of easily overlooked, but practically indispensable tasks:

- Truck drivers – needed to drive trucks to and from our storage facility to deliver machines to the data center. These volunteers also helped unload machines from the trucks and move them to the unboxing area.
- Unboxers – responsible for unboxing machines and the rack mount equipment, and loading them on the carts to send into the data center.
- Machine movers – responsible for transporting equipment from the nearby delivery point into the data center.
- Recycling team/clean-up crew – responsible for sorting all materials into appropriate recycling bins and cleaning shipping areas and the data center of any debris.

As word of this initiative spread through campus, and the number of volunteers rose, we also added two other unlikely groups of volunteers:

- Check-in assistants – responsible for handing out nametags and directing volunteers to areas in need of assistance.
- Food area help – responsible for serving breakfast and lunch for volunteers, as well as clean-up.

In all, some 120 volunteers helped deploy the remainder of the cluster.

Waste Removal

Every single compute node was provided by the vendor in an individual box. In addition to the server, and rack mounting equipment, each box also contained product manuals, a CD of firmware and diagnostic programs, a power cord, and two cable management trays. None of this extra material was used.[2] Given the number of systems involved in this cluster, managing the waste was a substantial task.

We recycled or reused nearly all of the extra materials we processed at the install site. Power cords and cable management trays were returned to the vendor for reuse. The shipping pallets were similarly collected, for reuse by Purdue's shipping and receiving group.

We arranged for recycling services to remove the large amounts of foam and cardboard generated as we unboxed these machines. This was all planned in advance and recycling services placed dumpsters at our location before the event started. Recycling services removed full dumpsters of foam and cardboard several times during our install. In total, some 660 lbs of packing foam, 6000 lbs of cardboard, as well as additional manuals and CDs, were recycled during the main installation.

[2]The vendor-supplied power cord was approximately 6 feet (2 meters) in length. Its use, or the use of the provided cable trays, would have caused airflow obstructions and would have presented a thermal hazard in our environment. However, it was more costly to have the vendor change the packing procedure for these systems than to just purchase shorter power cords ourselves.

Infrastructure Performance

May 5th installation activities started at 8:00 am, Eastern Daylight Time (EDT). By 11:45 AM, when a general lunch recess was called, only 80 compute systems remained to be mounted in equipment racks. Over two hundred systems had been deployed, the effects of which can be seen in Figure 1. Following a one-hour break, work resumed. By 1:30 pm, all systems had been racked and wired. By 3:00 pm, all systems had at least begun the OS deployment process, requiring no further human intervention. At that time 750 systems (out of 806 total) were reporting as available, and 1400 user-submitted batch jobs had already begun running on the compute cluster.

By noon the following day, all but six machines had been made available to the University research community. By 4:00 pm on May 6th, all available systems had been connected to the Open Science Grid [15], and had begun processing additional international work.

By moving the bulk of the software installs into the kickstart configuration (which was served by the

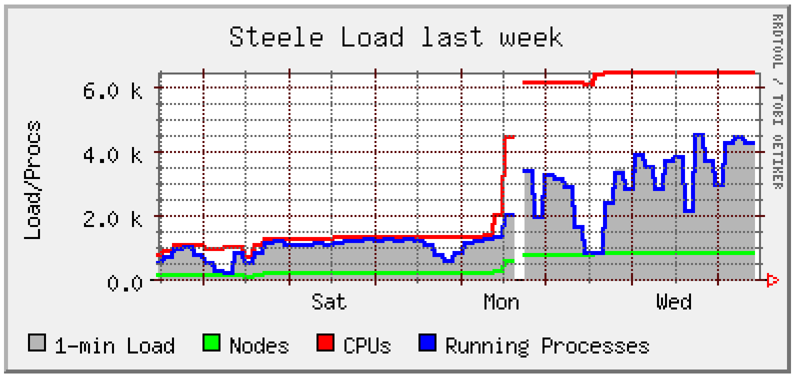
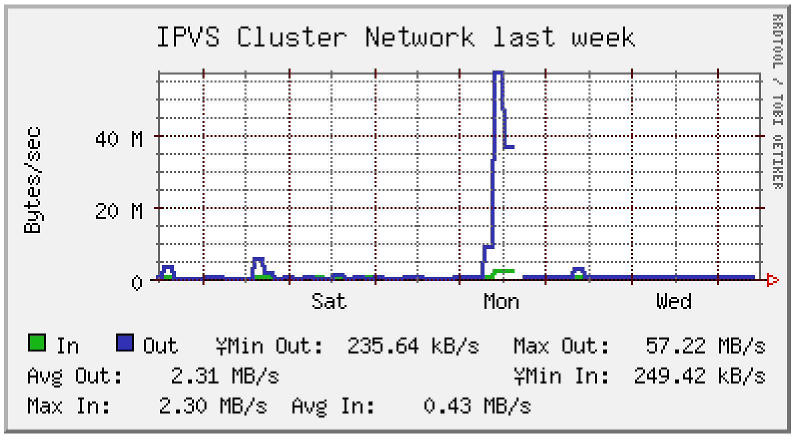


Figure 1: "Steele" cluster during deployment.



```
¥Min Out:  235.64 kB/s    Max Out:   57.22 MB/s
Avg Out:    2.31 MB/s               ¥Min In:  249.42 kB/s
Max In:    2.30 MB/s   Avg In:    0.43 MB/s
```

Figure 2: IPVS cluster network use.

IPVS cluster), our RedHat proxy server wasn't overly strained – its load average ranged from 0.1-3.0 throughout the day. For larger deployments, however, we may consider either increasing the install time or, more likely, adding additional proxy servers.

As seen in Figure 2, our IPVS cluster was sustaining data rates of about 57 MB/sec during the installation. While we did see traffic bursts as high as 390 MB/sec, these were very short-lived and did not adversely impact the deployment. We feel this infrastructure could sustain simultaneous installation to two to four times the number of machines, though if it did become a serious bottleneck we could simply add additional IPVS backend servers to the cluster.

We also knew a fair number of machines might be dead on arrival. We merely skipped those IP addresses in the deployment procedure, by running the CGI program to increment to the next available address. By the end of the day 56 had been skipped, mostly due to minor wiring problems or having been passed over for the command to boot via PXE, and never beginning the install process. These were easily corrected the following morning.

We did encounter one unexpected failure: the cluster monitoring suite we use, Ganglia [16], was initially unable to cope with the amount of memory in the "Steele" cluster. This issue had since been corrected in a later software release, so we elected to upgrade immediately to the newest version. As a side effect, cluster monitoring was briefly unavailable, as evidenced by the brief discontinuities in Figures 1 and 2.

Customer Acceptance and Benchmarking

One of the key groups behind the acquisition of the "Steele" cluster is Purdue's Network for Computational Nanotechnology (NCN), a community of researchers focused on simulation of nano-scale semiconductor devices. We began to study the performance of the fledgling cluster for this type of simulation the evening of May 5th, a matter of hours after the last of the compute nodes had been unboxed.

To evaluate the performances of the "Steele" cluster, a benchmark example was run on 16 to 6272 cores. For that purpose we used the quantum-mechanical nanoelectronic device simulator OMEN [17, 18]. This massively parallel software computes the current characteristics of nanotransistors as function of the input source, drain, and gate voltages. It has four levels of parallelism[3] and in its most inner loop two eigenvalue problems and a sparse linear system are solved. In a typical device simulation this happens more than hundred thousand times enabling the use of large computer clusters as "Steele."

Figure 3 shows the scaling properties of OMEN on the "Steele" cluster for the simulation of a silicon double-gate ultra-thin-body field-effect transistor designed according to the ITRS specifications for the 22nm technology node [19]. Only one bias point is computed with different parallelism schemes. The blue curve with crosses represents the case where the parallelization of OMEN is achieved with MPI only an no domain decomposition is applied to the transistor structure. Hence, the momentum and energy points are parallelized. To obtain the green curve with triangles we decompose the simulation domain on two cores with the help of a distributed memory sparse linear solver. Both these approaches require to launch as many MPI tasks as available cores.

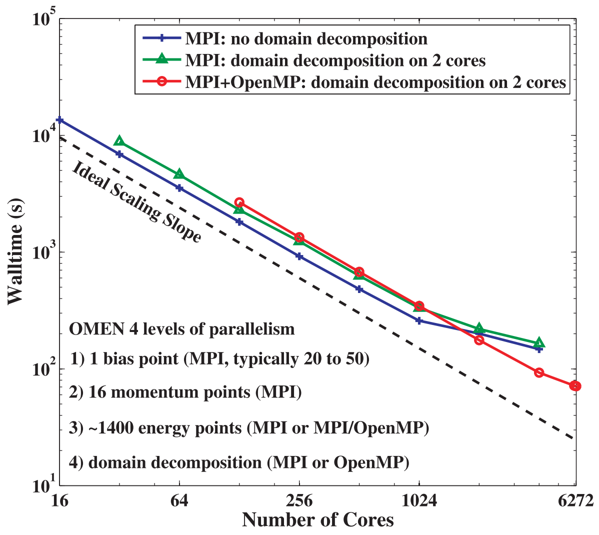


Figure 3: Application scaling on the "Steele" cluster.

An alternative is the implementation of a MPI-OpenMP hybrid where a single MPI task is started per node and eight threads are created within each node. The red curve with circles in Figure 3 illustrates this approach. Domain decomposition is realized on two cores using a shared memory sparse linear solver. For example, 784 MPI tasks are necessary to obtain the results on 6,272 cores. Note that the intra-node communication implied by the hybrid model helps reducing the overhead caused by MPI collective operations like MPI_Allreduce. Consequently, the red curve with circles exhibit almost no saturation of its scaling properties as compared to the two others.

Following these benchmarks, every research community involved in the acquisition of the "Steele" cluster resumed their normal usage patterns, after a total service outage of under nine days.[4] By May 8th, four days after 88% of the system was deployed, over four thousand batch jobs have been submitted to the new system. This has been both the largest single computer system in Purdue's history and the shortest delivery time for a high-performance scientific resource the University has seen.

[3]OMEN parallelizes across bias points, momentum points, energy points, and decomposition of the simulation domain.

[4]This counts both the time needed to remove older compute resources and to deploy all components of the new system.

Further Work

While we originally developed this deployment process to deploy large cluster systems, we have since begun to use it for nearly all system installations within our environment. During normal operation, the PXE boot system will bring up a menu by default, containing a list of all our common machine configurations (each of which load a custom ks.cfg configuration file into the appropriate OS installer). Several versions of RedHat Linux are supported, as is Debian Linux (through their FAI [20], or Fully Automatic Installation, system).

Our network deployment infrastructure is used daily to deploy individual machines. For such small quantities, we slightly modify our PXE configuration to provide the correct network configuration for the machine. A slight modification to our ks.cfg post-install script bypasses the USB-driven address assignment procedures if DHCP is not in use during system installation, so machines will proceed directly to cfengine configuration. Once this process is initiated, new servers are fully customized and ready for production work in approximately fifteen minutes.

The availability of this rapid deployment framework has also changed our response to problem mitigation. Rather than painstakingly correcting an installation issue on hundreds of systems (for example, changing the partition table on production compute servers), we merely schedule a brief downtime and reinstall the entire group of affected machines. We have reinstalled over 160 machines in under an hour using this process, and have been able to successfully return these systems to full service with a minimum of user-visible downtime.

Conclusions

The "Steele" installation was a huge success. We had planned for a full day for all of the physical labor and were expecting the OS installation and configuration pieces to run overnight. Considering that 200 nodes were online and processing jobs by lunch, with the rest of the functional systems available for cluster users by 3:00 pm, it's safe to say we met our target business-day turnaround time for this deployment.

Of particular concern was the handling of the large amounts of recycling materials. We had arranged for empty dumpsters for recycled goods to be delivered several times throughout the day. Thanks to the number of people that showed up to help, loading empty boxes and other debris was trivial. However, this just serves to underscore the need for proper site planning and coordination (in addition to the technical measures necessary) to prepare for a rapid deployment of this scale.

We believe mass server deployments, with hundreds of servers deployed in under a business day, are very achievable. Scaling the backend infrastructure to support these deployments can be done at low cost, using commonly-available (and largely open-source) technology, as we have demonstrated. With equivalent preparation, other institutions should be able to see similar rates of system installation. Eventually, we see this practice moving from the realm of IT "stunt" [21] to an accepted business process.

Acknowledgments

The authors would like to thank a number of individuals for their role in helping bring about the "Steele" cluster. The following people aided in designing, purchasing, transporting, installing, deploying, and organizing portions of this project: Hans Aagard, Samarth Agarwal, Lon Ahlen, Syed Ali, Billy Anderson, Chris Andrus, Bill Ansley, Kim Arnold, Laura Arns, Marty Barnes, Andrew Bean, Jeremy Benner, Larry Biehl, Bart Bishop, Brian Black, Dave Black, Carolyn Bogan, Darryl Borgonah, Kyle Bowen, Allen Braunsdorf, Mitch Breneman, Dean Brusnighan, Birgitte Brydso, Michael Budzik, Tom Bunton, Clyde Burghardt, Wayne Byarlay, John Campbell, Jere Carroll, Felix Chang Lee, Phil Cheeseman, Min Chen, Tammy Clark, Kevin Colby, Mark Copeland, Benjamin Cotton, Douglas Couch, Robby Crain, Rob Cumberland, Donna Cumberland, Charles DeLa, Kent DeLa Croix, Noah Diffenbaugh, Steve Dong, Randy Drake, Jason Dufair, Ed Dunn, Rudy Eigenman, David Eisert, Alex Elman, Ed Evans, Vicki Farnsworth, Shane Farnsworth, Greg Farr, Alan Farrester, Pat Finnegan, Mike Fisher, Greg Flint, Barbara Fossum, Joe Francisco, Bruce Fuller, Hugh Fultz, Sue Gibson, Michael Gillaspy, James Gonzales-Meisler, Peter Goreham, Thomas Hacker, Ben Haley, Andrew Hall, Dave Hancock, Mike Hardebeck, Steve Hare, Matthew Harvazinski, Jessica Harvey, Patricia Harvey, Amy Haston, Brian Heaton, Nathan Heck, Laura Helms, Randy Herban, Michael Hicks, Scott Hicks, Elizabett Hillery, Rita Hillman, Matthew Hodson, Geoffrey Holden, Steve Holmes, Ben Holmes, Ryu Hoon, Mark Houser, Andy Howard, Nelson Howell, Fengping Hu, Shaomin Hu, Ben Huao, Kay Hunt, Rick Irvine, Wen Jiang, Bill Jones, Jennifer Kapp, Julie Kercher-Updike, Rene Ketterer, David King, Alex Kingman, Gerhard Klimeck, Scott Ksander, Sravan Kumar Dheeraj Kapilavai, Bruce Lambert, Ronald Lehman, Pam Lewellen, Markus Lill, Matt Link, Bruce Loftis, Matt Luisier, Doug Magers, Rishabh Mall, Mike Marsh, Ashlie Martini, Jennifer Maxson, Gerry McCartney, urtis McGrew, Dwight McKay, Diana McKinney, Sean McLane, Michael McLennan, Cory Meadows, Saumitra Mehrotra, Carol Middleton, Pia Mikeal, Manuriti Minhas, Eric Mullen, Mike Myers, Max Myers, Mary Nemacheck, Norbert Neumeister, Ryan O'Neill, Mark Padgett, Hyun Park, Terry Patterson, Abhijeet Paul, Jeff Pherson, Jim Polley, Rajini Prabhu, Nirmal Prajapati, Rajib Rahman, Jeffrey Rattray, Byron Reed, Neil Reed, Meiqi Ren, Matthew Riehle, River Chuan Xiao, Jim

Robinson, Timothy Rogers, Brian Rose, Michele Rund, Chad Sailors, Juan Santos, Richard Schick, Jim Schmitz, Krysten Schneider, Jeff Schwab, Louis Scott, Bernie Seabolt, Dave Seamen, Jie Shen, Jason Sheridan, Corey Shields, Mike Shuey, Richard Simmons, Anna Sledd, Steve Sloan, Nicholas Smith, Randolph Smith, Preston Smith, Julie Smith, Carol Song, Carmen Springer, John Steele, Jamie Stevens, Jeff Stewart, Jimmy Stine, Dave Stogner, Claire Strodtbeck, Matt Sutherlin, Andrew Sydelko, Steve Tally, Floyd Taylor, Helen Terrell, Andy Thomas, Richard Thompson, Jared Thompson, Jenett Tillotson, Todd Turner, Kerry Tyler, David Umberger, Kristen Van Laere, Phyllis Veach, Greg Veldman, Joe Wade, Judy Wagner, Bill Walker, Richard Westerman, Joe White, Bill Whitson, Drue Whitworth, Guneshi Wickramaarachchi, Kent Wiesner, Chad Wilhelm, Ramon Williamson, Jackie Wilson, Greg Wilson, Nancy Wilson Head, Dan Winger, Bryan Worthington, Jon Wright, Dongbin Xiu, Haiying Xu, Alex Younts, Joel Zarate, and Lan Zhao.

Bibliography

[1] Naughton, Thomas, Stephen L. Scott, Brian Barrett, Jeff Squires, Andrew Lumsdaine, and Yung-Chin Fang, "The Penguin in the Pail – OSCAR Cluster Installation Tool," *Proceedings of SCI'02: Invited Session – Commodity, High Performance Cluster Computing Technologies and Application*, 2002.

[2] Papadopoulos, Philip M., Mason J. Katz, and Greg Bruno, "NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters," *Concurrency and Computation: Practice and Experience*, Vol. 15, Num. 7-8, pp. 707-725, December, 2001.

[3] Burgess, Mark, "Cfengine: A Site Configuration Engine," *USENIX Computing Systems*, Vol. 8, Num. 3, 1995.

[4] Desai, Narayan, Rick Bradshaw, Joey Hagedorn, and Cory Lueninghoener, "Directing Change Using Bcfg2," *20th Large Installation System Administration Conference (LISA '06)*, pp. 215-220, December, 2006.

[5] *Puppet*, http://reductivelabs.com/projects/puppet/ .

[6] *Kickstart*, http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/en-US/System_Administration_Guide/Kickstart_Installations.html .

[7] K. Amorin, *Solaris Jumpstart Automated Installation*, http://www.amorin.org/professional/jumpstart.php .

[8] *IPVS*, http://www.linuxvirtualserver.org/software/ipvs.html .

[9] *Preboot Execution Environment (PXE) Specification*, http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf .

[10] *Knoppix Live Linux Filesystem on CD*, http://www.knopper.net/knoppix/index-en.html .

[11] Agerwala, T., J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, "SP2 System Architecture," *IBM Systems Journal*, Vol. 34, Num. 2, 1995.

[12] *cfengine*, http://www.cfengine.org/ .

[13] *Linux HA (HighAvailability)*, http://www.linux-ha.org/ .

[14] *Squid web proxy/cache*, http://www.squid-cache.org/ .

[15] *Open Science Grid*, http://www.opensciencegrid.org/ .

[16] *Ganglia Monitoring System*, http://ganglia.info/ .

[17] Luisier, Mathieu and Gerhard Klimeck, "OMEN: An Atomistic and Full-Band Quantum Transport Simulator for Post-CMOS Nanodevices," *IEEE NANO 2008, 8th International Conference on Nanotechnology*, August, 2008.

[18] Luisier, Mathieu, A. Schenk, W. Fichtner, and Gerhard Klimeck, "Atomistic Simulation of Nanowire in the sp3d5s* Tight-Binding Formalism: From Boundary Conditions to Strain Calculations," *Physics Review B 74, 205323*, 2006.

[19] Luisier, Mathieu and Gerhard Klimeck, "Full-band and Atomistic Simulation of n- and p-doped Double-Gate MOSFETs for the 22nm Technology Node," *SISPAD 2008, 13th International Conference on Simulation of Semiconductor Processes and Devices*, September, 2008.

[20] *FAI – Fully Automatic Installation*, http://www.informatik.uni-koeln.de/fai/ .

[21] Hayes, Frank, "Frankly Speaking: Learning from IT Stunts," *Computerworld*, May, 2008.

[22] RedHat Enterprise Linux, http://www.redhat.com/rhel/

[23] RedHat Network, https://www.redhat.com/rhn/ .

ENAVis: Enterprise Network Activities Visualization

Qi Liao, Andrew Blaich, Aaron Striegel, and Douglas Thain – University of Notre Dame

ABSTRACT

With the prevalence of multi-user environments, it has become an increasingly challenging task to precisely identify *who* is doing *what* on an enterprise network. Current management systems that rely on inferring user identity and application usage via log files from routers and switches are not capable of accurately reporting and managing a large-scale network due to the coarseness of the collected data. We propose a system that utilizes finer-grained data in the form of local context, i.e., the precise user and application associated with a network connection. Through the use of dynamic correlation and graph modeling, we developed a visualization tool called *ENAVis* (Enterprise Network Activities Visualization). *ENAVis* aids a real-world administrator in allowing them to more efficiently manage and gain insight about the connectivity between *hosts*, *users*, and *applications* that is otherwise obfuscated, lost or not collected in systems currently deployed in an enterprise setting.

Introduction

Complex systems are hard to understand and visualize. The causes for this problem are due to the specific data not being available or the inability to correlate and present the data in a meaningful and understandable way. Additionally, the administrator faces an overwhelming amount of data to manage especially on large scale enterprise networks. Network connections ranging from a few hundred to several thousand are generated on a daily basis by each host. Tracking down precisely *who* (users) and *what* (applications) are responsible for the generation of this network connectivity is a non-trivial task. Administrators need a tool that allows them to sift through massive amounts of traffic logs in a visually appealing and interactive manner that encourages data exploration rather than hindering it.

Despite the abundant amount of data available, the coarseness of the data derived from point-to-point logging does not make it particularly useful. The current logging schemes such as NetFlow [1] data, provide activity details in terms of IP addresses and ports, but are unable to tell which users and what applications are running on the managed network. Since the identity of the traffic flow is important [2], and the users and applications are the essential components of the network, the identity should be associated with the users and applications in addition to the hosts.

It is necessary for the *context* of a connection, i.e., the user and application responsible for the network activity, to be known rather than simply *where* (address) it came from and went to. Existing solutions to this problem have involved tie-ins of network flow data and authentication systems such as Active Directory [3] and Kerberos [4, 5]. Critically, these existing logging systems are not geared towards real-world system administration. Network flow data will only detail the *where* of a connection, whereas an Active Directory and Kerberos tie-in can explain the *who*. A few visualization and data exploration tools [6, 7] that exist, primarily rely on chaining together network connections based on the flow data. However, multiple hop connections are typically obfuscated due to the nature of network flows; the level of detail supplied is traditionally limited to the IP addresses and port numbers involved.

Rather, a method to interactively explore the inter-relationships of the data so as to gain insight as to what is occurring as opposed to inferring, due to lack of log details or time to trace-back and locate the necessary information, is needed. For example, if an account on a network is compromised then it needs to be known what hosts that user account attempted to log into, along with the applications and programs they attempted to run, and files that may have been modified or touched. Knowing exactly *who* (users) and *what* (applications), not inferring from IP and port, at *both* sides of connections is of particular interest in policy compliance auditing. Being able to present all of this information in a single visual appeasing and manageable view would be a tremendous asset for network administrators.

To facilitate solving the above problems, we present *ENAVis* (Enterprise Network Activities Visualization). *ENAVis* is a tool for visualizing the network activities among hosts/domains, users and applications, which is possible through the gathering of *local context* information. *ENAVis* offers interesting, ready-to-use, and invaluable functions for monitoring, visualizing, exploring and investigating the activities on a network by real-world network administrators.

Through the use of a highly detailed local context data collection system spanning over 300 machines with a mixture of student, faculty and grid computing nodes on the University of Notre Dame's campus since April

2007, we have collected over 300 GB of raw data and developed *ENAVis* to allow an administrator to explore this informative data set.

With *ENAVis*, the administrator is presented with an array of connectivity graphs and statistics on how the network is being used. To assist the user in understanding the many possible visualization modes, we provide a novel *meta-visualization* which compactly represents and controls how data is represented. By adjusting the Host-User-Application (HUA) control, the user may easily expand, contract, and explore a very rich data space in a visually appealing and highly interactive manner. Figure 1 illustrates the *ENAVis* approach and how it ties into an enterprise network.

The key highlights of this paper include:

- *Data Collection*: The light-weight, easy-to-deploy monitoring agent, the *Monitor*, collects the missing yet important local context information (who, what, when, and where) associated with each network connection in an enterprise network at a very fine level of granularity.
- *Graph Model*: Our novel hierarchical graph representation of data in terms of domain/hosts, users, and applications (HUA) captures the dynamic relationship and interaction between machines and user applications.
- *Visualization*: An easy-to-use yet powerful graphical interface that makes exploration of large amounts of network connectivity interactive and manageable.

The rest of paper is organized as follows. In the next section we discuss the objectives of our tool, i.e., the design principles and desired functions. We emphasize the problems this paper targets and propose our solutions. We then talk about the design and implementation of the data collection system. Next, the graph model in terms of combinations of hosts/domains, users, and applications is presented. Then we examine several important cases to demonstrate how the visualization tool functions. The design and implementation of functional models of *ENAVis* are presented in the following section. The related work section compares our system with currently existing tools. Finally, we conclude and suggest future work.

Objectives

It is good practice for administrators to log the system events and network activities [8]. However, the large amount of data accumulated each day is difficult for human beings to understand and explore. Visualization is therefore an important topic in network and system adminstration since it eases the manual process of going through log data and correlate information and present it in a meaningful way. The objectives of *ENAVis* is to plot various combinations of the feature/attribute vectors in the log data and provide a customizable and interactive interface for human auditors to explore and investigate the activities that occurred on their networks. Most importantly, a unique inter-hosts/users/processes matching capability included in *ENAVis* provides the administrator with intuitive information on the dependant relationships, which may help many other important problems such as security tracing and fault localization.

Problem Statement and Solution

There are two problems which we tackle in this paper. First, there is a lack of tools and data to capture the user and application level of network activities.

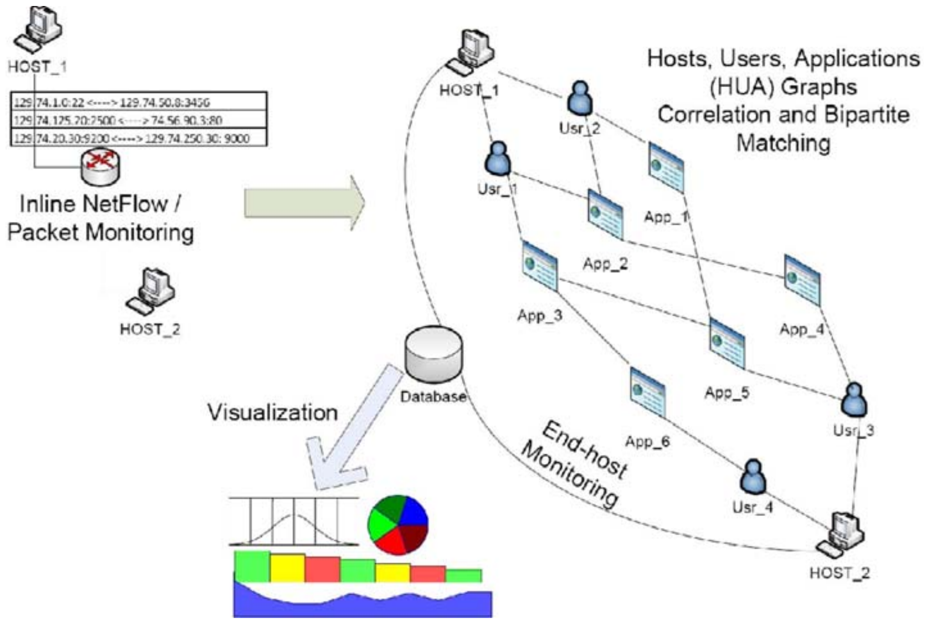


Figure 1: Inline (left) vs. end-host (right) monitoring scheme. End-host gatherers the missing local context (user, application, files, etc.) associated with each network connection.

Second, there is also a lack of tools to visualize and capture the inter-relationships of such data. They are discussed in more detail below.

In addressing the first problem, administrators do not usually lack for log data for security measurement [7]. However, administrators are facing a dilemma that on one side is an overwhelming amount of data, but on the other side many of these data are not at the level of detail they would like. Although there are tools to log network activities in either packet or flow format, there is no light-weight mechanism in current practice to monitor the network at a finer granularity than host-to-host. For example, the network IP addresses included in the packet header only means *locators* for the machines. It tells nothing about the *identities* of the end-users. On the other hand, the transport layer's port numbers are also less meaningful in determining the actual end-processes. While using deep packet inspection requires an understanding of all known protocols, it is still unknown which users and applications are sending those data.

Motivated by the observation that the end host has full *visibility* of the user's processes, our approach to the first problem is to deploy a simple agent on the end hosts to collect these missing local context data for each network connection. The agent is easy to deploy and lightweight in that it is purely written in a `bash` script that calls commonly available system tools such as `netstat` and `ps`. Through careful mapping between each TCP/UDP socket with the user ID and process ID, we associate users and applications with each network connection. The data is then sent securely from each host to a central database server for correlation, analysis and audit.

The second problem, independent of data collection mechanism, is how to understand and interpret the data. The natural question to ask is now that we have the data, how should we visualize it in a more intuitive manner? With the amount of workload on a busy system administrator, being able to quickly browse through the data, view summary statistics and charts, and interact with connectivity graphs can help them very much.

Visualization is the key to solve the second problem, which is the focus of this paper. It is commonly recognized that many of the human errors are due to the lack of understanding of their domain knowledge. A properly designed human-computer interaction can expedite data understanding and improve the exploration process. Our solution is to develop a powerful yet friendly graphic user interface that allows the network administrators to view their network activities at the user and application levels in addition to the topology created by the host connectivity . The design principles of our system are described in the next section.

Design Principles

The target of the system, namely what is to be achieved by this tool, is detailed below:

Know who, what, when and where (4W): The fundamental motivation of the system is for an administrator to know what is happening on their network, i.e., who (which users) are running what (applications) on where (which hosts) at when (what time). All information relevant to the connection context needs to be recorded.

Compute, generate, and trace heterogeneous graphs: In order to visualize the 4W aspects of the data, the tool needs to transform the raw data into an animated graph topology view. The graph is considered heterogenous because each node in the graph can be either a domain, host, user or application and edges are the network connections observed between them during a customizable time frame. Based on user events (such as clicking/dragging a node, applying filtering rules, and filtering number of hops to view from the highlighted node), the graph is instantly regenerated to reflect the changes. Figure 2 shows an example of such a graph. The bipartite matching (pairwise connections between nodes, users, and applications) simplifies the viewing and tracing of the network connectivity relationships among the nodes. Various graph algorithms [9] can be applied to produce interesting paths/cycles based on user activities.

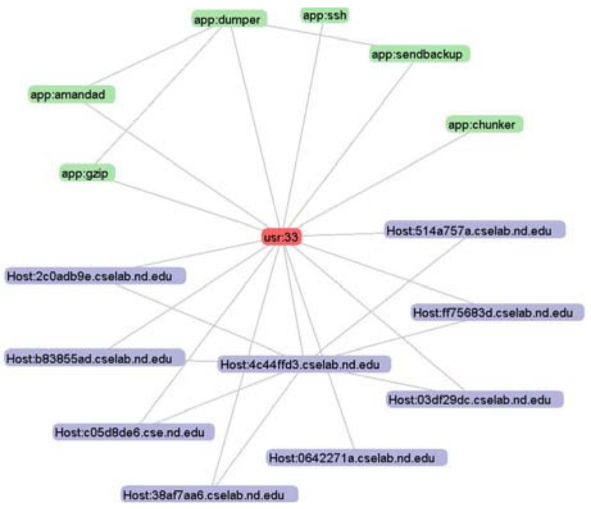

app:dumper
app:ssh
app:sendbackup
app:amandad
app:chunker
app:gzip
usr:33
Host:514a757a.cselab.nd.edu
Host:2c0adb9e.cselab.nd.edu
Host:ff75683d.cselab.nd.edu
Host:b83855ad.cselab.nd.edu
Host:4c44ffd3.cselab.nd.edu
Host:03df29dc.cselab.nd.edu
Host:c05d8de6.cse.nd.edu
Host:0642271a.cselab.nd.edu
Host:38af7aa6.cselab.nd.edu

Figure 2: An example heterogeneous graph generated by *ENAVis* contains host, user, and application nodes.

Investigate interactively: Although understanding data and recognizing the patterns among it through visualization techniques such as plotted charts and graphs is important, another important feature designed for the tool is the ability to explore the data *interactively*. Through only a few mouse operations, the administrator is able to make queries to the database, DNS, and LDAP servers for more detailed information, analogous to "please tell me more about this."

Plot charts and report summary statistics: The visualization tool should have the capability to plot charts based on time, host, user, and application information:

- line chart: useful for viewing number of connections for selected domains, hosts, users, and applications.
- pie chart: useful for determining the percentage that each host/user/application contributes to the total traffic generated.
- scatter plot: useful to see the distribution of connectivity points with possible combination of x/y coordinates such as IP addresses, port numbers, user IDs, and time.

The tool would also be able to provide summary statistics based on the daily log data, such as the top and average hosts, users or applications making the most number of connections, and to produce need-attention reports on demand for an administrator's review.

Make it simple, efficient and customizable: Ideally, the tool should be simple yet powerful, usable for real-world administrators.

- *Simplicity*: The tool must be easy to use even for first-time users. Exploring and viewing network activities should consist of only moderate mouse clicks.
- *Efficiency*: Despite the large amount of data available, if the tool responds too slow it will reduce the user's experience. We admit it is a challenging task to, upon user's request, query the database, download the files, plot charts, and generate animated graphs, while keeping efficient use of the available memory with large data sets.
- *Customization*: While most users will not need to modify the base set of views, the ability to customize via a modular viewer is a powerful feature. Ideally, users would be able to customize their configuration and build an environment in which they are most interested (ex. Top 10 Applications, Current Connectivity of Human Resource (HR) Users, Status of Grid Compute Nodes, etc.).

Consider future extensibility: One potential extension for the tool is to analyze the underlying data by applying various data mining and machine learning techniques. For example, building trees to classify network events, or building clusters to group similar user behaviors, and identifying anomaly based on the model built. The extension for data mining and anomaly detection is an ongoing work and will be included in future releases.

Data Collection

This section describes the type of data we have been collecting and gives an overview of the entire system. It defines and introduces local context. As discussed earlier in the Objectives Section, the first problem we are trying to solve is how to collect the missing context, i.e., to capture the user and application level of network activities (4W). The system we propose ties the user and application identities into the enterprise network management by utilizing existing tools (netstat, ps, lsof), which together build a hierarchical gathering of local context related to network connectivity.

System Overview

The data gathering component utilizes commonly available tools in order to take advantage of development robustness and administrator familiarity. The tools should augment the existing data significantly, i.e., not just another method to report *IPflows* or *SNMP* data.

A natural fit for these criterion is the netstat tool, in essence the equivalent of *whois* for network connectivity. Moreover, netstat can be coupled with other tools such as the process table via ps (linking process

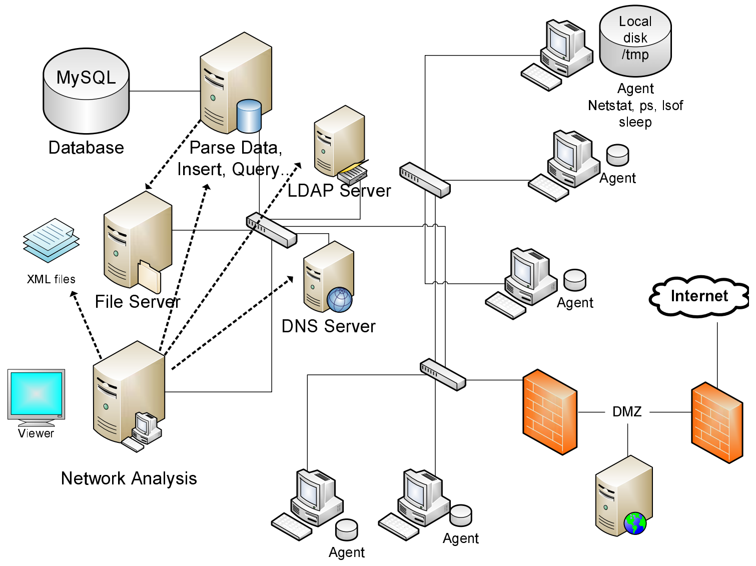


Figure 3: Overview of the system's architecture shows the monitoring, collecting, analyzing and visualizing of the local context from the connections made by users and applications within an enterprise network.

ID to the application and arguments) and the open file handles via lsof (linking the application to files and an alternative method for linking the application to connection). Each of the tools or equivalent is present by default on most major operating systems and each of the tools runs with minimal computational cost.

Figure 3 shows an overview of the data gathering and analyzing architecture. Each host employs the monitoring agent whose purpose is to periodically poll the tools and push the locally buffered data to the repository for future analysis. The administrator can then retrieve the data from the repository (or repositories) for the purpose of analysis and forensics from a single vantage point replete with local context. The *local context* is defined as the information fully detailing a network connection (protocol, src/dst IP/port), user, application, application arguments, and network-related file accesses. The lightweight nature of the system comes from the fact that it provides local *context* with regards to the presence of connectivity (network and files), not the *content* passed in the connectivity itself (data payloads, packet headers, etc.).

A Hierarchy for Gathering Local Context

We now briefly describe the three major tools used in our data gathering system, what each supplies, and how the supplied information can be fused together to provide a complete view of the local context associated with each network connection. Conceptually, one can view the data available from the tools and their fused data in terms of tiers. In the base tier, Tier 1 (simple local context), only netstat data is analyzed. The next tier, Tier 2 (enhanced local context), enhances the local context of netstat to offer increased application information as well as the process tree. The final tier, Tier 3 (complete local context), offers insight regarding potential information flow (what files a connected process is touching) and a more precise identification of the application (exact path, libraries, etc.). An example of the result of the fusion of the data from these three tools as stored in our database is shown in Table 1.[1]

Tier One (netstat)

netstat [10], is the most important command utilized to capture each instance of network connectivity occurring on the monitored system. In comparison to the standard rules in the firewall, netstat provides similar information with regards to the connection tuple (protocol, source IP, destination IP, source port, destination port). The State field can be any of the twelve

[1]For privacy purpose, host names are hashed and IP addresses are mapped by using prefix-preserving anonymization technique.

Host	Proto	Local IP	Local Port	Foreign IP	Foreign Port	STATE
32dfdffb	tcp	180.83.70.53	33318	180.83.46.242	636	ESTABLISHED
57e0a268	tcp	180.83.70.224	9230	162.203.142.116	50942	ESTABLISHED
321fc626	tcp	180.83.193.184	43825	180.83.46.242	636	ESTABLISHED
cf58df4b	tcp	180.83.21.235	22	242.86.74.143	46688	ESTABLISHED
bb326ee6	tcp	180.83.21.98	39493	180.83.46.242	389	ESTABLISHED
ad8a26cf	tcp	180.83.41.162	9679	17.11.56.128	0	LISTEN
3a677f01	udp	17.11.56.128	40423	17.11.56.128	0	–

Start	Stop	UID	GID	i_node	PID	PPID	Direction	Application
1178116633	1197926231	104092	40	6875664	30525	30520	1	firefox-bin
1177746632	1197926196	108172	40	29494600	16114	4325	2	condor_starter
1178046253	1178047203	119100	40	12618336	15863	1	1	mozilla-bin
1190653418	1198003091	0	0	5981424	27669	3669	1	sshd:
1190607911	1197926168	105273	40	8121972	12901	24139	1	vim
1177381883	1177381889	108172	40	25103156	10615	4264	0	condor_starter
1177425206	1177867251	0	42	26116251	4365	3771	0	gdm-binary

Path	Args
/usr/lib64/firefox-1.5.0.10/firefox-bin	-UILocale en-US
/afs/nd.edu/user37/condor/software/i386_rhel30/sbin/condor_starter	-f macbeth.rcac.purdue.edu
/usr/lib/mozilla-seamonkey-1.0.8/mozilla-bin	-UILocale en-US
/usr/sbin/sshd	[accepted]
/usr/bin/vim	exercise1.c
/afs/nd.edu/user37/condor/software/i386_rhel30/sbin/condor_starter	-f bach.helios.nd.edu
/usr/bin/gdm-binary	-nodaemon

Table 1: Sample network connectivity data from the fusion of netstat, ps and lsof (Host names and network addresses are anonymized). Among the fields, *HosT*, *Proto*, *Local IP*, *Local Port*, *Foreign IP*, *Foreign Port*, *State*, *i_node*, *UID*, and *PID* are from netstat; *Application*, *Args*, *GID*, and *PPID* are from ps; *Path* is from lsof; *Direction* is deduced from previous *Listen* state; *Start* and *Stop* are from diff.

values such as SYN_SENT/RECV, FIN_WAIT, etc., but we focus on the LISTEN and ESTABLISHED state for TCP connections.

Tier Two (netstat+ps)

ps is the second tier command that is used to supplement the information from netstat. It provides a list of all current running processes. Although the *-p* flag in netstat provides important information such as the program ID/name responsible for each socket, it does not provide the whole picture. Through another lightweight tool, the ps [11] command, not only is the application name make available, but also the arguments provided to the application can be retrieved. We note that while lsof tool with *-i* option provides similar information as netstat and ps supply, lsof is not available everywhere and less stable than netstat and ps.

Tier Three (netstat+ps+lsof)

The optional lsof [12] command lists each open file on the current host and provides the third tier of information. By extracting the PID and UID from netstat and/or ps, a linkage can now be made to what files are being accessed for the PID responsible for a network connection. With the help of lsof, a more accurate picture of the application itself can be provided, as noted by the absolute application path (not just the executed command), the libraries, and files touched by the application.

The most interesting aspect of lsof is the discernment of an application's location. From a policy management standpoint, centrally served (ex. NFS/AFS mount) or validated local versions (ex. MD5, SHA1 hash) can reduce the ambiguity associated with applications. The notion of classifying according to *application location* can offer an additional mechanism for extracting characteristics such as versions of applications. In a broad sense, one could view applications as existing in one of three forms, user local (local directory or user path), machine local (root-level install, ex. /usr/bin), and enterprise served (root-level mounted). The file accesses of the applications noted by lsof can also be categorized in a similar manner.

Host Config Info

When the agent component initializes for the first time, it collects an array of system-wide information that is sent back to the central administration server. The information collected includes:

- Current System Time
- Host name and OS version (i.e., uname -a)
- Snapshot of /etc/passwd and /etc/group
- List of iptable rules (i.e., iptables -L)
- Network Interface Parameters (i.e., ifconfig -a)
- Hardware info (i.e., /proc/cpuinfo, /proc/meminfo, /proc/uptime, /proc/version, etc.)
- Tool Versions (i.e., netstat --version, ps --version, lsof -v, etc.)
- Any other information that administrators would like to collect.

Implementation

The data collection agent was implemented as a bash script that calls UNIX commands netstat, ps, lsof, and diff periodically. The benefit of implementing the agent as a script is its immediate deployability without any special changes to the network or hosts. MySQL server is set up on an dual-core Opteron box running Solaris 10 with two 400 GB disks. A parsing program written in Java that is used to parse the collected raw data from each host and is inserted into the database using the Java Database Connectivity (JDBC) interface. The structure of the database is composed of a set of tables, each of which stores the output from each of the tools as described earlier in this section.

The data collection agent is deployed on 300+ machines throughout our campus. The machines are a mix of CSE faculties and students office computers, scientific grid computing nodes, and engineering lab machines. The goal was to capture various characteristics ranging from manual human interaction to batch job oriented network connection styles. We have been running data collection over one year since April 2007 with a database size of 300 GB. Although the current state of the deployed agent utilizes only the Linux version of these tools, Solaris and Mac OS X versions have been developed and tested as well, and a native Windows agent is under development.

Concerning the cost of agent deployment, the average CPU usage of the agents observed on hosts in our engineering computer labs where students may log on via console or ssh peaks at four or five percent only when the agent is awakened to call the netstat, ps, lsof and diff. The empirical data suggests that the configurable sampling rate of five seconds is a good balance of granularity of logging and overhead. The memory usage is bound by the usage of those standard UNIX tools. Concerning the diff output size, lsof has the largest volume followed by netstat and ps. As stated earlier, the usage of lsof is optional due to its relatively high expense when compared with netstat and ps. Overall, the average total data size of each host per day is 2.8 MB, or in other words less than 1 GB for each monitored host per year. Moderate disk space requirement allows for one common 500 GB disk to store all data for an entire year on a 500-host network. Since the agents push out the data every 15 minutes in our setting, the total bandwidth consumption for collecting such data is only 120 Kb/s for a monitoring scale of 500 hosts (all hosts within the 120 Kb/s), and therefore the network bandwidth overhead is negligible.

While an event-based model appears more appealing, a bash script that only uses standard UNIX commands is adopted for fast and easy deployment without any modification to the kernel or recompilation for different architectures. On the other hand, our novel usage of diff output (by comparing previous and current calls) achieves the event-based model to some extent because only the difference is recorded, not all data. The difference can be interpreted as the beginning of a new connection/activity or the end of an existing connnection/activity. It is understood that the data collected in this polling scheme may not be perfect and could miss some transient events such as TCP connection state changes. The script is also not the most efficient way of logging compared to a compiled binary program. One of the purposes of the system is to invoke thoughts on what type of data should be collected and how ready-to-deploy and widely available tools can achieve this. It is also possible that we combine the reports from the end hosts with the NetFlow data if we want more accuracy in connection time, direction, packet size, etc. The full visibility at the end hosts provides a richer context (in terms of users and applications) of network connectivity that is not readily available from inline monitoring.

Network Connectivity Graphs

In this section, we lay out the theoretical foundation for the graph representations of the data we collected. We make a unique contribution using a heterogenous graph model that involves mappings between hosts, users and applications (HUA). The interesting graph model can have applications in the area of enterprise network management, security, auditing, problem debugging and fault localization. Figure 4 shows one of the graphs of the network viewed through *ENAVis*.

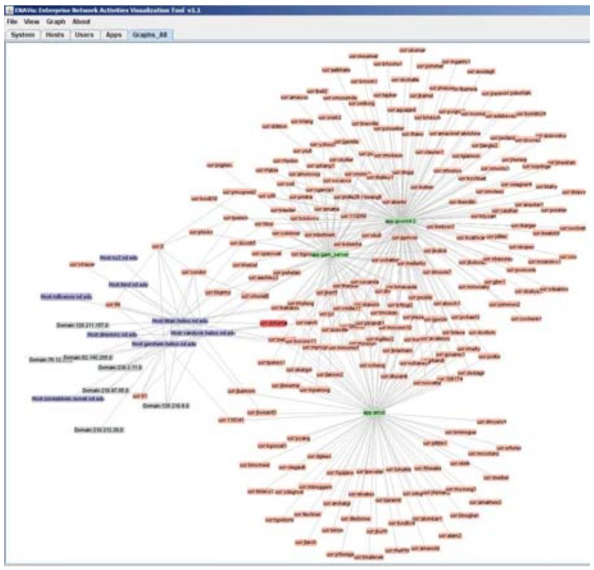

ENAVis: Enterprise Network Activities Visualization Tool v1.1
File View Graph About
System Hosts Users Apps Graphs_All

Figure 4: A graph view of the network connectivity data, a feature included in the *ENAVis* tool. Exploration starts from various operations on a selected node, which can be either host, user or application.

User and Application Chaining

The motivation for doing user and application level matching comes from the question: *what are the foreign applications and users behind the other side of the connection*? It is of particular interest as the traditional packet analysis is not of any usefulness in knowing the identity of applications or users. With our system, the identity (user/application) of *both* sides of the end-to-end connection can be linked together assuming both hosts are monitored.

In its simplest form, a bipartite matching is found if an established connection recorded on Host A with src_A and dst_B matches another established connection record on Host B with src_B and dst_A within the same time frame. The time frame can be from a single hour to several days depending on the granularity requirement. It is easy to see that by going through the n records of all established connections and bucket-sorting those records into a destination-based lookup hashtable $table_{dst}$ will take linear time. Going through $table_{dst}$ and building a second source-based lookup hashtable $table_{src}$ as described in Algorithm 1 will also take linear time. To create the connection chains, we iterate through all n records; each step requires two lookups in $table_{dst}$ and $table_{src}$, which takes constant time. The number of records in $table_{src}$ to be fused and outputted are at most n in the worst case if all n recorded connections occurred between monitored hosts. Therefore, the complexity of the above chaining algorithm is $O(n)$, where n is the number of recorded established connections.

```
Input: conns (records of established connections
              within a time window)
Output: a bipartite matching of connections
  Bipartite_Matching(conns)
foreach record in conns do
  bucket sort by $src_i$ into $table_{dst}$;
end
foreach key in $table_{dst}$ do
  make $table_{src}$ whose keys are $dst_j$ and values are
    original connection records with $src_j$ and $dst_j$;
end
foreach record in conns do
  if $table_{dst}$ contains key ($dst_i$) AND
     $table_{src}$ also contains key ($src_i$) then
    find bipartite matching;
    output the fusion of $conns_i$ and $table_{src}$'s
      records;
  end
end
```

Algorithm 1: Connection Chaining.

Table 2 shows an example of such connection chaining after the fusion of the log data uploaded by the agents. Each new connection chaining record begins with the *start* and *stop* time of each connection and is further divided into the left and the right part. The top part is the *local identity* in terms of host name, IP/Port pair, user, and application associated with the connection. Similarly, the bottom part is the *foreign identity* in the same format. Before, at one end of the connection (say at server side), the identity of *who* connecting to the server is vaguely inferred from the

IP/Port pair (assuming only user A can use that client machine). Now, the identity of *who* is connecting to a host can be precisely known from the bipartite matching (no longer inferred from the IP/Port). Which *user* and what *application* are revealed at *both sides* of connection. This is useful in evaluating the effectiveness of the enforcement of the existing policy on the enterprise network.

Heterogenous Graph Model

Once we have collected and matched enhanced connectivity information, the next step is to visualize the network connections in the form of a graph. However, while we have a rich pool of data containing the entirety of host, user, and application connectivity, it is not desirable to view all the data at once. For instance, we may only want to view how users are interacting or perhaps would like visualize which application mix is being executed and by what hosts. To that end, we have created a novel control tool based on three core node *characteristics*, i.e., hosts, users, and applications (HUA), as illustrated in Figure 5. At the top, we have H denoting the *host* level chaining. This is also the most common scenario, in which all the connectivity between physical end-host machines is constructed. At the lower left and right, we have U and A, denoting the *user* and *application* level chaining; this is useful when we want to quickly know which users or applications have been communicating with each other.

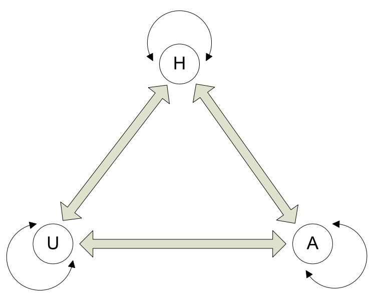


Figure 5: A meta graph illustrating the various combinations of states (H, U, A, HU, HA, UA, HUA) for modeling our network connectivity graphs. H, U, and A stands for *H*osts, *U*sers, and *A*pplications respectively.

The interesting exploration continues when we consider the various combination of the states shown

Start	Stop	Location	Host	IP/Port(proto)	User	Application
1177527137	1177527148	Local	211fba9b	180.83.12.112/631(tcp)	0	cupsd
		Remote	3a9d336a	180.83.12.178/34406(tcp)	97392	gnome-pdf-view
1177543303	1177543309	Local	211fba9b	180.83.12.112/631(tcp)	0	cupsd
		Remote	06baa7ef	180.83.12.85/35775(tcp)	92362	gedit
1177448975	1177449026	Local	f464cee2	180.83.183.147/54427(tcp)	105464	parrot
		Remote	c9c6e734	180.83.159.14/9094(tcp)	108172	chirp_server
1177391778	1177391807	Local	0642271a	180.83.12.72/40096(tcp)	33	dumper1
		Remote	38af7aa6	180.83.12.241/33084(tcp)	33	amandad
1177392075	1177392151	Local	0642271a	180.83.12.72/40211(tcp)	33	dumper3
		Remote	2c0adb9e	180.83.12.172/38429(tcp)	33	gzip
1177392075	1177392151	Local	0642271a	180.83.12.72/40212(tcp)	33	dumper3
		Remote	2c0adb9e	180.83.12.172/53342(tcp)	33	sendbackup
1177515292	1177515299	Local	b83855ad	77.46.16.81/36019(tcp)	317	httpd
		Remote	b83855ad	77.46.16.81/1521(tcp)	27	oracletestdb
1177610657	1177611222	Local	c9c6e732	180.83.159.108/9094(tcp)	108172	chirp_server
		Remote	ad8a26cf	180.83.41.162/49857(tcp)	102744	condor_exec.e
1177610633	1177610638	Local	c9c6e733	180.83.159.135/9710(tcp)	108172	condor_schedd
		Remote	ad8a26cf	180.83.41.162/9788(tcp)	108172	condor_startd
1177625404	1177625765	Local	c9c6e733	180.83.159.135/9314(tcp)	102744	condor_shadow
		Remote	ff75683d	180.83.12.132/9868(tcp)	108172	condor_starter
1177548953	1177548992	Local	af8f7bb2	180.83.12.155/34479(tcp)	97464	ssh
		Remote	633bfecc	180.83.12.167/22(tcp)	0	sshd:root
1177459056	1177459112	Local	06baa7ef	180.83.12.85/34739(tcp)	92362	gedit
		Remote	211fba9b	180.83.12.112/631(tcp)	0	cupsd
1177541462	1177541473	Local	06baa7ef	180.83.12.85/35714(tcp)	0	ssh
		Remote	633bfecc	180.83.12.167/22(tcp)	0	sshd: root

Table 2: An output example of bipartite matching. Not only are the IP Address and Port known for each established connection, but also the User and Process identity at both ends of the connections now become known (host names and network addresses are anonymized here).

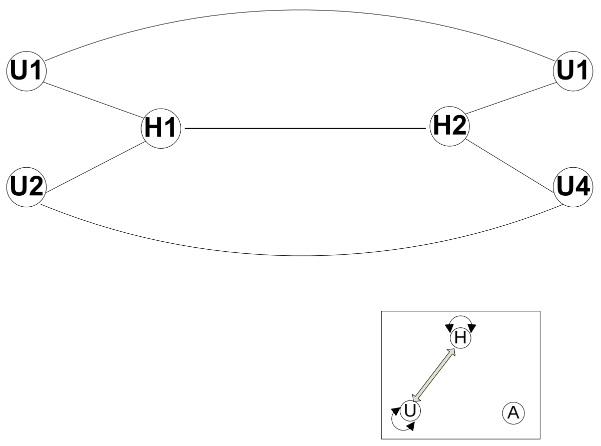


Figure 6: A HU graph representation showing user 1 (U1) on host 1 and 2 (H1, H2) are talking to each other. Similarly, user 2 (U2) on H1 and user 4 (U4) on H2 are talking to each other. The user-level bipartite matching resulting a shortcut path that forms a simple cycle to distinguish multiuser connectivity.

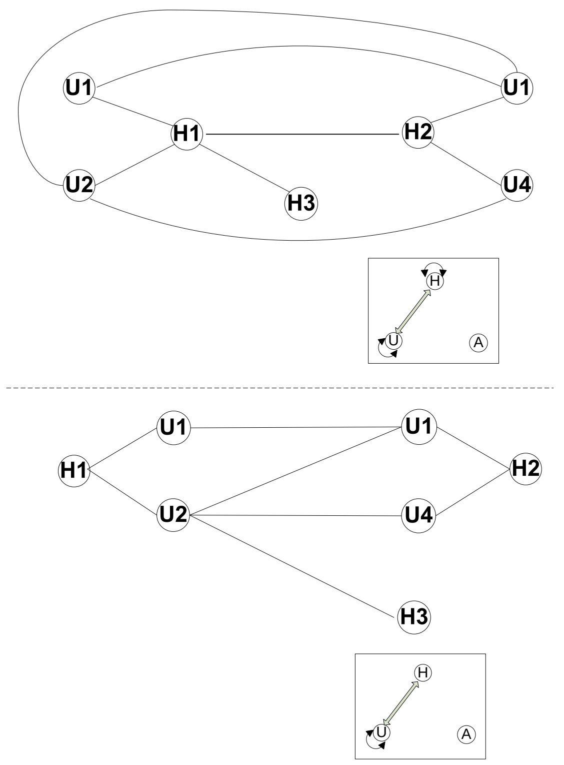


Figure 7: Suppose host 3 (H3) is an external host not running the agent and therefore no mapping between users on H1 and H3. The top graph has ambiguity between U1 and U2 that connecting to H3. An alternative HU graph representation can be constructed by removing edge between host nodes.

in Figure 5, namely a heterogenous graph containing the hosts, users, and applications (HUA). We can imagine a 4D space, where the time, host, user and application interact with each other. We briefly discuss these graphs below.

H: At a higher layer, we have the host connectivity, basically denoted by traditional IP/port pairs among servers and clients. Using H only is analogous to a connectivity view offered by NetFlow data.

U: At the middle layer, we have the user connectivity, in which we can observe the connectivity relationships among the users. Because multiple users can log onto the same machine and a single user can log onto multiple machines, by treating an enterprise user (no matter how many physical hosts they have logged on) as one single entity node, we are able to observe the overall network activities among users.

A: At the bottom layer, we have the application connectivity, in which we can observe the connectivity relationships among applications. A simple example would be which browsers are interacting on my intranet web server (i.e., Firefox 2.0, Internet Explorer 7, etc.) without worrying about user-agent spoofing. Similarly, what applications (and their versions) are checking out licenses from my license server?

HU: The first mix-mode is the interaction between users and hosts. As we said earlier, a user can log on multiple hosts and a host has multiple users simultaneously logged in. There are two options for constructing such HU graphs. First, we simply merge the H and U graphs. This 'glue' process is done by constructing an edge between the user and the host only if that user has made at least one connection on that host. A simple graph example is illustrated in Figure 6. Notice there is no ambiguity in *who* causes the traffic between host 1 and 2 because the connectivity forms a simple cycle (i.e., no vertex is traversed twice) that covers both vertices H1 and H2. For example, we know user 2 on host 1 has connections with user 4 on host 2, but user 1 cannot have connections with user 4 because a simple cycle is not formed. Note that this is only true when both host 1 and 2 are running the agents. If there is an external domain not under our control, there can be ambiguity in this representation, as shown in Figure 7. Therefore, the second graph representation of the data is constructed by simply removing the edge between hosts, taking the observation that there must *a user* associated with *each* connection on a host. Host nodes can be reached from the user nodes.

HA: Similar to HU, hosts and applications can be used to construct a connectivity graph when users are of less concern. Constructing HA graphs is similar to constructing HU graphs.

UA: The concept of location of physical hosts becomes less relevant as the real players on the network are the users and applications. In this case, we can temporarily filter out H and only leave UA because we are more interested in who (users) and what (applications) are running on the network. UA graphs is therefore a perfect choice. Constructing such graphs are similar to HU and HA.

HUA: Lastly, building hosts, users and applications into one graph provides the most comprehensive view as we show in later case studies. Constructing such a graph is just merging H, U, A graphs by using user nodes as the 'glue' for host and application nodes, i.e., an edge is drawn between a host and a user and between a user and an application if the user on that host has made at least one connection using that application. An edge connecting two application nodes represents the network connectivity between their respective users on two end hosts.

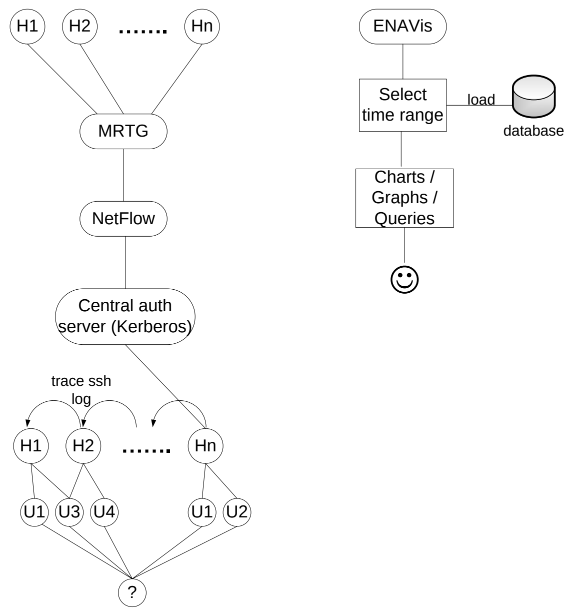


Figure 8: On the left is an example of problem tracing carried out by a sys admin by hopping through central authentication server log and local hosts ssh logs. On the right we show that admin does not have to deal with the scale of a distributed system. The investigation is a quick, convenient, fixed-step process with mouse-click driven exploration.

Application Discussion and Case Studies

We discuss several cases scenarios in which *ENAVis* can be helpful in local network management. The graphical exploration reduces the tedious, error-prone nature of log checking and mapping down to a few mouse clicks, which makes administrators' life much easier. With the capability of correlating hosts, users and applications through interacting with HUA graphs and straightforward statistical charts offered by *ENAVis*, the investigation carried out by the system and network adminstration can be confined to *O(1)* steps and does not have to hop through *O(n)* hosts in scale of a distributed system. Figure 8 illustrates the benefit of *ENAVis* visualization.

We will study several cases in detail with supporting graphs and data from using the tool. These scenarios are

- user and application-level policy compliance check;
- find the source of network bandwidth slowness;
- investigate and cleanup after user account compromise;

Scenario 1: Policy Compliance

The management needs to know whether their employees have complied with the company's network usage policy with regards to finance information compliance. Specifically, the administrator is requested to provide a report of whether the mechanisms are adequate for enforcing the current policy. For this case study, consider a financial intranet server whose access policy is defined such that only authorized users can access or even see the financial system. To that end, a set of host-based firewall rules are put in place on the finance server (*finance.nd.edu*) with restrictions to the hosts of authorized finance personnel (*concert.cse.nd.edu*, *striegel*) on the company campus.

Current Approach: First, the admin checks that the firewall rules (IP/port) settings are correct for the finance server through the application of a policy rule visualization tool such as [13]. Once the rules are validated, the administrator checks the *ipfilter* log and NetFlow log data to ensure that only authorized hosts accessed the server. Upon only seeing authorized hosts on the list (*concert*), the admin concludes that the policy is sound and not violated.

***ENAVis* Approach**: Unfortunately, the earlier approach is only sufficient if the host to user mapping stays consistent, i.e., only user *striegel* uses the host *concert.cse.nd.edu*. If host to user mapping is dynamic or unclear, the notion of host as identity quickly breaks down (see Figure 9). Suppose in the same environment that *ssh* connectivity was enabled on the network. In the scenario of Figure 9, an unauthorized user *qliao* connects from *IrishFB.nd.edu* with X11 forwarding to *concert.cse.nd.edu* and launches an instance of *firefox* to access the finance web server. In a multiuser environment, where multiple users are logged onto the same machine and make network connections, other tools have no way of differentiating those connections because the connections all have the same source IP. Similarly, the legitimate user *striegel* may carelessly connect from a *Starbucks* shop to his office desktop *concert* in order to access a financial account just for convenience. Neither of these two cases is desirable and is a violation of the policy because the original intent of the policy was that anyone not part of the finance department should not be able to access the finance host.

The tiered graph created with *ENAVis* includes nodes representing hosts, users and applications, taking advantage of our data which records every UID and PID associated with each network socket created. Since each connection tuple now is expanded to be {time, proto, src_ip/port, dst_ip/port, usr, app}, we have finer granularity on the policy control on the user and application level in addition to the host level, which can be clearly seen from the *ENAVis* graph (Figure 9). The admin is able to find the problem which is not offered by other tools, namely the unintentional configuration of *concert* with no ssh restriction causes the violation of the policy.

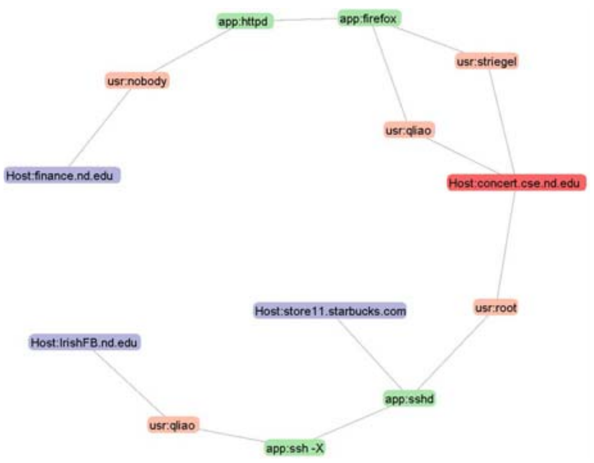


Figure 9: An *ENVAVis* HUA graph captures two possible host IP ACL policy violations caused by the unintentional configuration on host *concert* without ssh restriction.

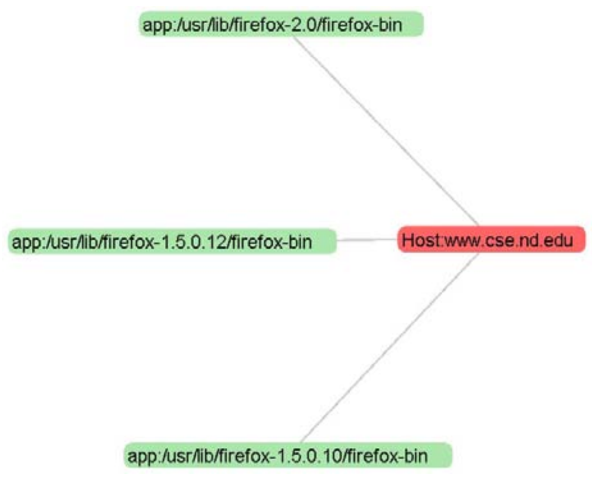

app:/usr/lib/firefox-2.0/firefox-bin
app:/usr/lib/firefox-1.5.0.12/firefox-bin
Host:www.cse.nd.edu
app:/usr/lib/firefox-1.5.0.10/firefox-bin

Figure 10: A simple example showing policy compliance control on application versions for vulnerability avoidance and license management.

In addition to policy compliance checks at the host and user level, another side benefit is to check the policy compliance at the application level. Consider the case when the admin wants to make sure only the most up-to-date version of applications are approved for use on the network (see Figure 10 for an example). There have been known vulnerabilities in earlier version (1.5.x) of firefox and the policy states users must use the properly patched version. A simple HA graph would reveal any non-compliance with this policy by looking at the applications connecting to the web server. In addition to vulnerability control, it is also useful for *license management*. Usually, the organization buys a fixed amount of licenses from the software vendors. The license server should only check out a license to legitimate users and newest version of the application software. Our tool makes it possible to track this type of compliance as well.

Scenario 2: Network Bandwidth Slow

In this scenario, users file a case report to the system administrator complaining about the network being slow.

Current Approach: The admin pulls the MRTG data through SNMP queries to the routers, and determines everything looks fine. The routers only has 30% of traffic load. Since the bandwidth slowness was reported five hours ago, the admin searches the NetFlow log data trying to locate the problem host. Finally, the admin locks down two problem hosts: one is a graduate student's desktop and the other is a machine in a lab.

For the graduate student's machine, the admin is pretty sure that student is the cause, thus she sends out a warning message to that student for a suspected violation of the network usage policy and if the user does not comply his network port will be shut down. For the machine in the computer lab, unfortunately it is in a multi-user environment. The admin has to decide who has been on it during the problem time period. She spends much of her time searching the ssh logs and correlating the logging information with a central box such as a Kerberos server trying to find who was logged in during that time. Finally, the admin narrows the search down to 10 users that have been on the system during the two hours of heavy use.

Needless to say the process is tedious. Imagine if the admin has 1000 machines and an administrator has to log onto each one to look at the ssh log, it will waste tremendous amounts of time. The process is also less fruitful because the admin cannot determine if the increased network connectivity is due to a legitimate reason (i.e., research experiments, etc.) or illegitimate purpose (i.e., illegal file sharing, etc.).

***ENAVis* Approach**: The admin loads the most recent data collected by the agents into the visualizer and has a quick plot on the number of connections across her network (Figure 11). The admin clearly sees a spike between January 10, 2008, time 9:00 and 16:00, which matches the network slowness complaints that users had reported. The increase in network activities is mainly contributed to an enterprise user. The admin selects each cluster from the drop-down menu and quickly narrows the search down to an abnormally busy host: *clapton.cse.nd.edu* (Figure 12). By simply clicking on the host name, a pie chart is automatically plotted to reflect the top *users* on the host *clapton.cse.nd.edu*, as illustrated in Figure 13. On the left pane, all users

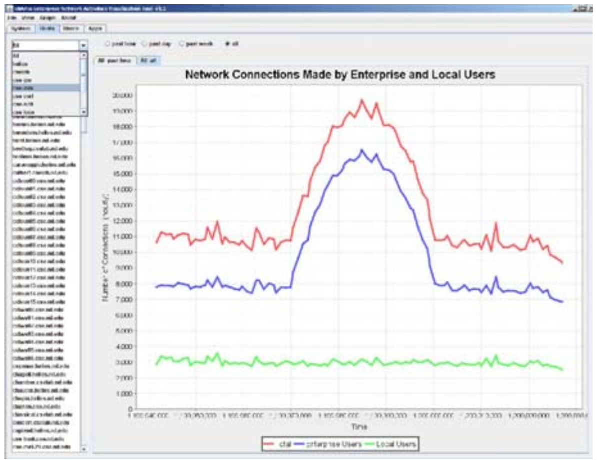

Network Connections Made by Enterprise and Local Users
Number of Connections (Hourly)
Time
ctal
Enterprise Users
Local Users

Figure 11: Number of hourly network connections (top line) separated by *enterprise* (middle line) and *local users* (bottom line) on the monitored network. The admin sees a spike in network activities.

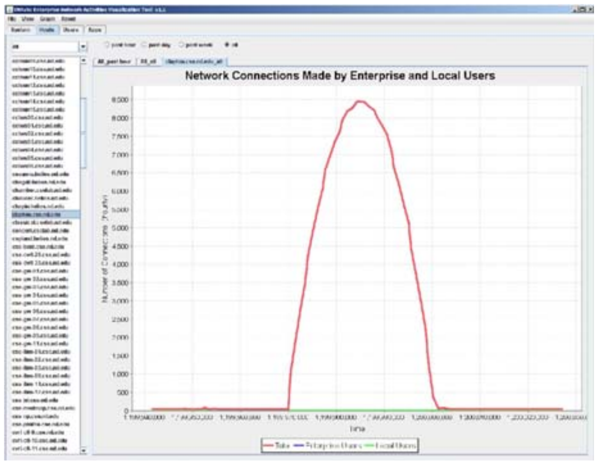
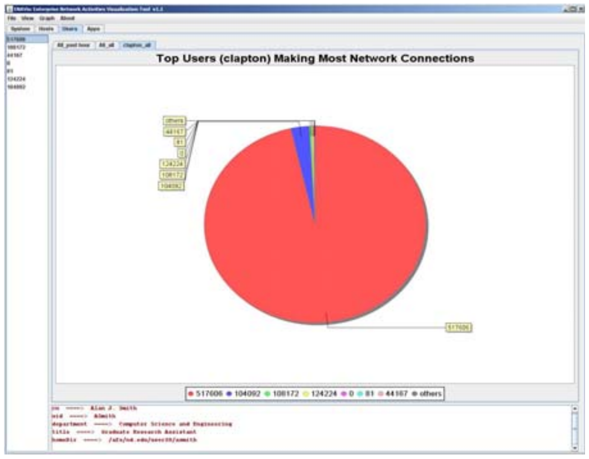

Network Connections Made by Enterprise and Local Users
Number of Connections (Hourly)
Time
Total
Enterprise Users
Local Users

Figure 12: After examining the charts on a few clusters, the problem source is pinned down to one host's (clapton) abnormally high network activities.


Top Users (clapton) Making Most Network Connections
others
44167
81
0
124224
108172
104092
517606
517606 104092 108172 124224 0 81 44167 others

```
cn ----> Alan J. Smith
uid ----> Asmith
department ----> Computer Science and Engineering
title ----> Graduate Research Assistant
homeDir ----> /afs/[illegible].edu/[illegible]/asmith
```

Figure 13: The left pane of the *User* tab includes a complete list (ordered by the magnitude of network connections) of users logged on the selected hosts/clusters in the *Host* tab. In this case, user 517606 has the top activities among all users on the host clapton. LDAP lookup on UID is displayed on the bottom pane.

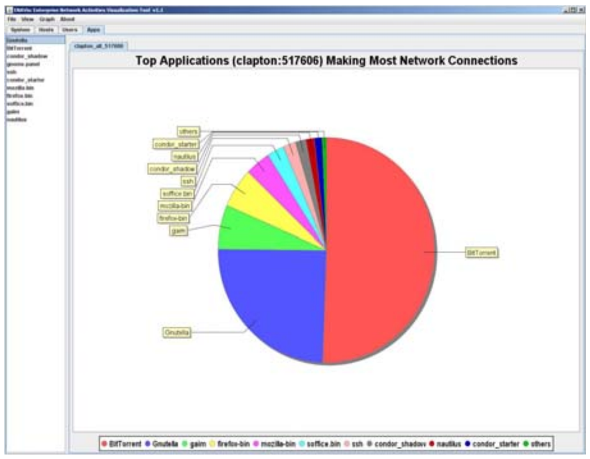

Top Applications (clapton:517606) Making Most Network Connections
others
condor_starter
nautilus
condor_shadow
ssh
soffice.bin
mozilla-bin
firefox-bin
gaim
BitTorrent
Gnutella
BitTorrent Gnutella gaim firefox-bin mozilla-bin soffice.bin ssh condor_shadow nautilus condor_starter others

Figure 14: View the category of top *applications* run by a problem user (517606) that making most network connections. File sharing applications such as BitTorrent and Gnutella occupy three quarters of the total connections on host clapton.

logged on that host that have made at least one network connection are listed in decreasing order.

The admin clicks on the problem user ID, two things happen. First, the tool automatically performs an LDAP lookup (it will check a local cache first to avoid excessively hitting the LDAP server) and displays the user information on the bottom of the graph (i.e., the user's first/last name, department affiliation, user name and AFS directory, etc.). Second, it automatically plots a pie chart (Figure 14) showing the top *applications* that user had used to make network connections. It is straightforward to see from the chart that the file sharing program BitTorrent and Gnutella constitute the top two applications that the user *asmith* had used. By now the problem has been traced to the source and the necessary action as dictated by the compliance policy will take place. All the admin does is less than ten mouse clicks. The graphical visualization and automation makes the admin quickly pin down the problem source without a tedious manual search process. The central correlation of the *user* and *application* information is the key.

Scenario 3: Cleanup After Compromise

A phishing email pretending to be from the IT department claims they are updating the system and require all users to send in their passwords, or their accounts will be suspended. A naive user believes this scam and therefore his password is suspected to have been compromised.

Current Approach: The system administrator needs to find out which hosts the compromised user account has used. Have those hosts been compromised as well? What applications did that user invoke? What data files did this user account touch during the past two weeks since the user revealed his password? The admin must make sure the student/faculty's sensitive information and intellectual property was not leaked from the network. In order to do this, the admin checks a centralized server such as an Active Directory or Kerberos 5's log file. Fortunately, the log file is still there, and the admin can then manually search and find all hosts that user has been trying to log into via the ssh pluggable authentication modules (PAM). The admin logs into each machine and makes sure they are clean. However, the admin has no idea what files have been read/modified or been sent out to external hosts. The admin also does not know what applications have been run by that user account because the data is not available.

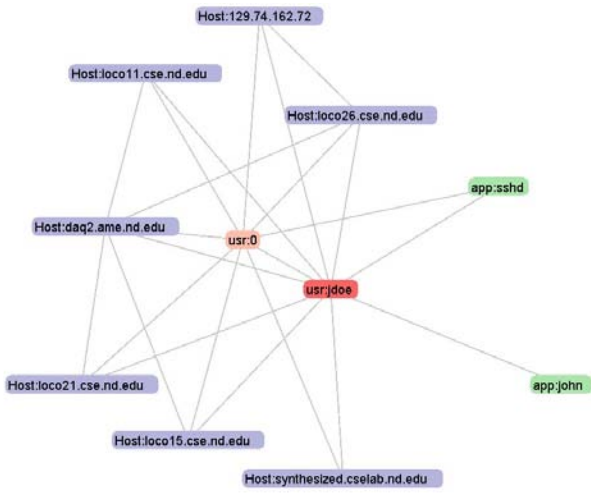


Figure 15: The HUA network graph reveals a highlighted user (jdoe) has logged on seven machines via ssh and has used the application John the Ripper to crack password files on those machines.

***ENAVis* Approach**: The admin simply generates a network graph by selecting the HUA from the graph menu. The admin highlights the problem user node (*jdoe*) (as in Figure 15). It is straightforward to see which hosts the user has touched during the time frame and what applications the user used. The file access information logged by lsof is not available to other tools, neither in a centralized authentication server nor in the normal end-host's access logs. Although we do not normally plot the lsof data, each file accessed by that user is kept in the master database. Therefore, a single query would reveal all files that user ID has touched among all the hosts. In this case, a visual graph is very helpful to see what hosts and users that a compromised user account has contacted and which applications it has attempted to launch. This helps expedite significantly such an investigation should it occur.

Other Functionalities

There are a few other potential uses of the tool, which we briefly cover here. For example, network fault localization. By comparing and contrast the difference between a working set and a problem set of network connectivity graphs, a system administrator would be able to detect the possible causes of network faults. Another example would be forensic auditing. The detailed user activities recorded in database may provide evidence when it is needed by some government agency.

Data mining is another potential use of our tool to detect possible anomalies in the network by invoking data mining and machine learning algorithms built into the tool (future work). The tool automatically colors nodes (hosts, users, applications) based on the clusters. Various classifiers kick in to evaluate the risk scores of the network events. It then generates a report that needs the attention or possible action by the management team.

ENAVis: The Visualization Tool

This section describes the implementation of functions of each module in the viewer[2] and how they can be used to explore the local context of monitored networks. The visualization tool was implemented using Java. The plotting functions utilized *JFreeChart* [14] and the graph animation was build on top of *Prefuse* [15], both are free open-source Java libraries.

Time Selection and System Message

As mentioned earlier, the tool should provide a quick summary of the past history, a time window defined by the user, and provide extensive reports on statistics of the hosts, users, and applications. The *start* and *stop* time of an investigation can either be specified as the command line arguments or simply selected from a GUI calendar object within the tool's interface. The tool then scans through the local disk to check if it already has the data files for the specified time range. If not, those files will be downloaded on demand in the form of either XML or comma-separated files at the user's choice. The "Update" button causes the tool to synchronize with the data file server.

Number of Connections Made By Users

The "Hosts" tab, shown in Figure 11, presents an overview picture on the number of network connections made by either *enterprise* users or the *local* users. On the left pane, administrator can select a set of predefined physical clusters (or all nodes in the network) from a combo box (drop-down list), which in turn propagates a complete list of monitored hosts within the cluster, where the user can select each specific host to view.

[2]More information and code available at http://netscale.cse.nd.edu/LockDown .

After selecting which host, cluster, or all monitored nodes, the user can further specify the time granularity of investigation by selecting one of the four radio buttons on top, i.e., "past hour", "past day", "past week", or "all". Based on the selection, a line chart is automatically refreshed to reflect the change. The different colors of lines, as indicated in the legend box at the bottom, indicates whether it is made by enterprise users or the local users. The differentiation of *enterprise* and *local* users is through querying the LDAP servers. The query results are cached locally to ensure any future lookup on the same UID will not hit the LDAP server.

Host Configuration

Each host configuration information can be displayed by right-clicking the host name in the list, and choose "configuration" from the pop-up menu. This triggers a query against the database host information table that return a complete list of a host's system statuses collected by the agent. The information currently stored for each host is: OS type and version, patch level, up time, local user and group info, firewall (iptable) rules, ethernet and network addresses for each network interface, hardware (CPU/mem/disk) information and versions of various system tools; as described in the Data Collection part of the paper. This functionality gives a quick and handy way for the system administrator to view each system status within the tool without requiring logging into each machine separately.

Alternative Data View

While graphical visualization is great, the option of being able to examine the raw data is always handy just in case the investigator needs to. Therefore, a table view is provided within the tool interface to display all raw connectivity records.

Users

Based on the various combination of hosts/clusters and time frames selected in the "Hosts" tab, the "Users" tab shows a vivid percentage summary as a pie chart for the top *n* users that are making the largest number of network connections. Figure 13 is one screenshot. The bottom frame shows a summary of user information: first/last name, netID, AFS home directory, department affiliation and job title, which are pulled from the enterprise LDAP server with the similar cache scheme described earlier.

Applications

The investigation flow continues in "Apps" tab (Figure 14), which shows a classification on the top applications run by a specific user. The pie chart tells a network administrator *what* is running on his network, e.g., are they mainly web browsers, email clients, printers, office software, or *condor* batch jobs, etc.? Presumably, different users have different behaviors in choosing personalized applications for network accesses. This is especially interesting for data mining and anomaly detection.

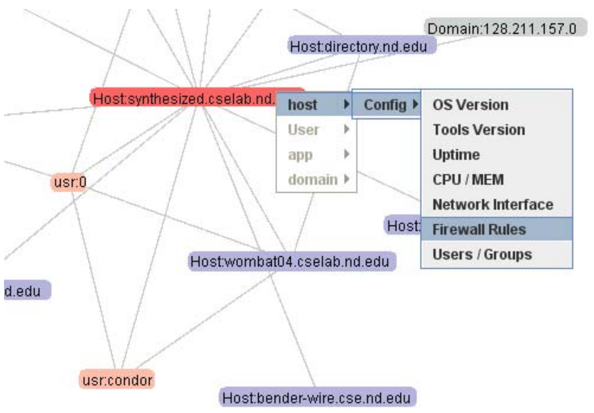


Figure 16: Popup provides detail-on-demand. In order to facilitate "please tell me more" function, popup event is implemented to display node properties by querying the database.

Connectivity Graphs

The network graphs, a significant feature included in the viewer, are supported by the open source *Prefuse* [15] library. We perform the bipartite matching on the nodes and transform the data into the GraphML format [16]. A heterogenous graph view of the network connectivity graph is presented earlier in Figure 4. On the right pane is a control tool set that can adjust the animation of the nodes interacting with each other by setting drag force, spring length, etc. The view can also zoom-in/out and be dragged around.

The connectivity filter allows the viewer to display only the number of hops from the question node. The hop count can be increased to give an extended view of the connected components in a larger chaining path. The node filter allows an investigator to select an arbitrary node (host/domain, user, or application) in the graph from a combo box rather than trying to locate a node in the graph itself.

The interactive feature is introduced to each generated graph through right-clicking on nodes. The menu is enable/disabled based on the context of the node type. For example, for a host node shown in Figure 16, one can query the database on-demand by simply select an item in the popup menu.

Related Work

Broadly speaking, the network monitoring and analysis can be categorized into two models. In the first type, in-network devices record and collect data using tools such as `tcpdump` or Cisco's NetFlow [1] profiling. The other type is end-host monitoring using an agent mechanism. The end-host monitoring approach has the advantage of being able to see more information than inline monitoring since it has full visibility of the network activities occurring on each host. We adopted the latter model for our data collection system.

sFlow [17] uses agents on switches/routers to log packets and send the logs to a central collector for analyzing. However, the traffic monitoring is at the packet level, thus missing the local context information for each connection. Another network traffic data visualizer is Multi Router Traffic Grapher (MRTG) [18] that monitors router traffic in a graphical form based on SNMP-enabled devices. There are also a few other visual analyzers based around using NetFlow's data. ISIS [6] is a tool that visualizes temporal relationships among network flow data by using a timeline and event plot. By plotting time in combinations with IPs, ISIS trys to find correlations between events to aid investigations regarding network intrusion. NVisionIP and VisFlowConnect-IP [7] have also been developed to visualize NetFlow data. As stated earlier, the key weakness of NetFlow data is the missing *user* and *application* information, which we posit is critical for enterprise network management.

Visualization techniques have been applied to view *static* data, such as distributed firewall rules to detect potential conflicts or anomalies. PolicyVis [13] is a visualization tool for inspecting firewall rules. It helps detect policy anomalies by plotting IP addresses and port numbers specified by the firewall rules in a 2D space and looking for overlap. We are in line with one of their motivations that visual inspection can be useful in understanding the otherwise complicated relationships among this form of data. Instead of visualizing policy rules, we visualize the *dynamic* data, which is the actual network activities made by users' applications. The visual analysis done on the empirical data is a substantial and necessary supplement to the static rules inspection as a proof of correctness to the policy rules.

Beyond the analysis of network data, various clean slate efforts have attempted to bring identity into the network flow. Among the re-architecturing attempts in the enterprise network, SANE [19] and Ethane [20] take a drastic approach in that instead of using a traditional layered approach, a single protection layer governs all connectivity within the enterprise. The enforcement of enterprise-wide security policies is done at the link layer. User authentication to a centralized server and switch-level source routes are mandatory to access services and end hosts. Within a SANE enterprise, IP address are not used for identification, location, or routing.

Finally, out of the various related works, perhaps the works closest to ours are those of [21] and [22]. In [21], the authors propose capturing the inter-dependencies among network components in 'Leslie graphs,' based on the original dependency work of Lamport. The "black-box" approach relies on the correlation of observed network traffic to infer system dependencies. The agents in their system called *AND* perform temporal correlation of the packets sent and received by the hosts; where the central server engine performs Bayesian inference from the reports generated by the agents. While these works mainly focus on computing the dependency graphs for fault localization (i.e., debugging the location of network failure or sluggish performance), our system focuses on the lightweight aspects of information gathering and how to visualize not only connectivity but, the context of the connectivity itself. In short, while these tools help to locate dependency-related performance problems at the host-level in a theoretical sense, ENAVis provides a robust platform for exploring and visualizing the connectivity data for a much wider assortment of security and performance-related issues.

Conclusion

It is desirable, yet difficult, to know exactly *who* and *what* is running on an enterprise network. In current network architecture, the identity of *user* and *application* in network flows is inferred from a packet's *content* (i.e., IP addresses and port numbers) rather than directly from the *context* (user processes) that actually make those connections.

In this paper, we describe a network local context data collection system and *ENAVis*, an Enterprise Network Activities Visualization and analysis tool. In addition to the regular analysis functions provided similarly by NetFlow and packet monitoring tools, *ENAVis* offers interesting new features of visual analysis on the user's and application's level. Connectivity graphs in combinations of hosts, users and applications capture the dynamic interactions among these essential components in the network, and provide an interactive exploration of the network connection log data. Future work is planned to incorporate data mining techniques into the tool to aid in automatic analysis of the data.

Acknowledgements

The authors would like to thank Curt Freeland, faculty specialist at CSE department of University of Notre Dame, for helping deploying the system. Special thanks to Adam Moskowitz and the anonymous reviewers for their insightful comments.

This work was supported in part by the National Science Foundation (CNS-03-47392, CNS-05-49087) as well as a Sun Academic Excellence Grant (AEG) (EDUD-7824-080234-US).

Author Biographies

Qi Liao is a Ph.D. student at the Computer Science & Engineering department of the University of Notre Dame. His current research interests include computer security, network management, data mining and economic applications on networks and security. He received his master degree in computer science and engineering (*MSCSE*) from the University of Notre Dame, Indiana. Qi graduated with a *B.S.* and Departmental Distinction in *Computer Science* from Hartwick College, New York, with minor concentration in *Mathematics*. He is a member of *Kappa Mu*

Epsilon and *Upsilon Pi Epsilon*. Reach him at qliao@nd.edu .

Andrew Blaich is a Ph.D student at the University of Notre Dame's Computer Science and Engineering Department. His research interests are focused on computer security and networking; with current work being done on network management. He received his B.S. and M.S. in Computer Engineering from Villanova University. Andrew can be reached at ablaich@nd.edu .

Dr. Aaron Striegel is currently an assistant professor in the Department of Computer Science & Engineering at the University of Notre Dame. He received his Ph.D. in December 2002 in Computer Engineering at Iowa State University under the direction of Dr. G. Manimaran. His research interests include networking (bandwidth conservation, QoS), computer security, grid computing, and real-time systems. During his tenure as a student at Iowa State, he worked for various companies in research and development that included Sun Microsystems, Architecture Technology Corporation, and Emerson Process. He has received research and equipment funding from NSF, DARPA, Sun Microsystems, Hewlett Packard, Architecture Technology Corporation, and Intel. Dr. Striegel was the recipient of an NSF CAREER award in 2004. Dr. Striegel can be reached at striegel@nd.edu .

Douglas Thain is Assistant Professor of Computer Science and Engineering at the University of Notre Dame. His research interests focus on harnessing large scale computing systems such as clusters, clouds, and grids to attack large problems in science and engineering. Dr. Thain received the Ph.D. from the University of Wisconsin in 2004. He can be reached at dthain@nd.edu .

Bibliography

[1] Cisco Systems, "Introduction to Cisco IOS NetFlow – A Technical Overview (White Paper)," October, 2007, http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html .

[2] Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture," *RFC 4423*, May, 2006.

[3] Microsoft, *Planning, Implementing, and Maintaining a Microsoft Windows Server 2003 Active Directory Infrasture,* Microsoft Press, 2003.

[4] Neuman, C., T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," *RFC 4120*, July, 2005.

[5] MIT, "Kerberos: The Network Authentication Protocol," 2008, http://web.mit.edu/Kerberos/ .

[6] Pham, D., J. Gerth, M. Lee, A. Paepcke, and T. Winograd, "Visual Analysis of Network Flow Data with Timelines and Event Plots," *Workshop on Visualization for Computer Security (VizSEC)*, Sacramento, CA, pp. 85-99, October 29, 2007.

[7] Yurcik, W., "Visualizing Netflows for Security at Line Speed: The SIFT Tool Suite," *19th Large Installation System Administration Conference (LISA '05)*, San Diego, CA, p. 16. December 4-9, 2005.

[8] Takada, T. and H. Koike, "MieLog: A Highly Interactive Visual Log Browser Using Information Visualization and Statistical Analysis," *Proceedings of the 16th USENIX Conference on System Administration (LISA '02)*, Philadelphia, PA, pp. 133-144, November 3-8, 2002.

[9] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, second edition, MIT Press and McGraw-Hill, 2001.

[10] netstat(8), *Linux Programmer's Manual.*

[11] ps(1), *Linux User's Manual.*

[12] Abell, V., *LiSt Open Files (lsof), Open-Source, UNIX Administrative Tool*, ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/ .

[13] Tran, T., E. Al-Shaer, and R. Boutaba, "PolicyVis: Firewall Security Policy Visualization and Inspection," *21st Large Installation System Administration Conference (LISA '07)*, Dallas, TX, pp. 1-16, November 11-16, 2007.

[14] JFreeChart, "Free Java Chart Library," http://www.jfree.org/jfreechart/ .

[15] Prefuse, "The Prefuse Visualization Toolkit," http://prefuse.org/ .

[16] GraphML, "The graphml File Format," http://graphml.graphdrawing.org/ .

[17] sFlow, "Traffic Monitoring Using sFlow," 2003, http://www.sflow.org/sFlowOverview.pdf .

[18] Oetiker, T., "MRTG – The Multi Router Traffic Grapher," *12th Systems Administration Conference (LISA '98)*, Boston, MA, pp. 141-147, December 6-11, 1998.

[19] Casado, M., T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks," *15th USENIX Security Symposium*, Vancouver, Canada, p. 10, July, 2006.

[20] Casado, M., M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," *Proceedings of ACM SIGCOMM*, Kyoto, Japan, pp. 1-12, 2007.

[21] Bahl, P., P. Barham, R. Black, R. Chandra, M. Goldszmidt, R. Isaacs, S. Kandula, L. Li, J. MacCormick, D. A. Maltz, R. Mortier, M. Wawrzoniak, and M. Zhang, "Discovering Dependencies for Network Management," *ACM SIGCOMM 5th Workshop on Hot Topics in Networks (Hotnets-V)*, Irvine, California, pp. 97-102, November 29 and 30, 2006.

[22] Bahl, P., R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards Highly Reliable Enterprise Network Services via Inference of Multi-Level Dependencies." *ACM SIGCOMM Computer Communication Review*, Vol. 37, Num. 4, pp. 13-24, 2007.

Fast, Cheap, and In Control: A Step Towards Pain Free Security!

Sandeep Bhatt, Cat Okita, and Prasad Rao – Hewlett-Packard

ABSTRACT

We hypothesize that it is possible to obtain significant gains in operational efficiency through the application of simple analysis techniques to firewall rule sets. This paper describes our experiences with a firewall analysis tool and metrics that we have designed and used to help manage large production rule sets. Firewall rule sets typically become increasingly unwieldy over time. It is common for firewalls to have hundreds, or even thousands, of rules. Not surprisingly, administrators have a hard time keeping track of how the rules interact with each other, resulting in many partially effective or completely ineffective rules, and unpredictable behavior. Our tool can be used to identify these problematic rules. Further, given two rule sets, our tool produces a comprehensive list of the traffic that is only permitted or denied by one rule set, rather than both. As such, we can compare the existing rule set with a second rule set containing the proposed changes. The administrator can then visually check if the difference in traffic patterns corresponds to what he or she intended in proposing the changes. Additionally our tool collects various metrics that help the administrator to gauge the 'health' of the firewall. The tool is designed to be extensible to multiple vendor products.

Introduction

Securing the increasingly complex and highly networked environments of today is a challenging and frustrating task. Between compliance demands, such as Sarbanes-Oxley and PCI, new applications and services, and increased end-user awareness of security issues, it is a challenge to maintain security in any environment. As the complexity of an environment increases, the complexity of the configurations required to secure access to the environment increases, as does the amount of time required to maintain, update and validate the configurations. Changes to the security configurations start to produce unexpected side effects, and often result in a reluctance to make any changes at all!

Similarly, the amount of time required to debug obvious access issues increases dramatically with the size and complexity of the configurations involved – and skyrockets when multiple groups are involved in debugging any single issue. Configuration issues which do not result in immediately obvious issues such as permitting additional access that either fails to be noticed (or is so useful that nobody wants to mention it!) are often missed, and are hard to find without extensive, repeated effort on the part of an experienced system administrator (nowadays affectionately referred to as a 'resource').

Migration between vendors, or versions of security devices is also challenging – there is no standard configuration language for security devices like firewalls, and most vendors do not provide migration tools between their own OS releases, let alone from the OSes of other vendors.

Compounding the above issues, since the security of an environment is often a case of proving a negative, there is a definite need for straightforward, easily understood metrics that can be used to describe the state of a given configuration. Alas, while there are many tools that can be used to secure an environment – firewalls, intrusion detection/prevention devices, virus scanners, et al. – there is a dearth of simple, multi-platform tools that can be used to analyze and measure existing installations.

Problem

In this work, we have elected to focus on ways to improve firewall rule set management, which many security managers have identified as an ongoing challenge. Typical operational issues include how to determine the effect of adding or removing a firewall rule, clean up messy firewall rule sets and debug firewall rule related issues; other challenges include reporting the ongoing status of firewall operations in an easy to evaluate format.

The increasing commoditization of firewall management through outsourced and managed services, as well as decreasing amount of time or skill (or both) available to manage firewalls in house has also led to an increase in "cargo cult" style firewall operations, and a "once in, never out" rule set management.

Methodology

Our goal was to produce a high value, fast, lightweight tool that can be used to improve firewall rule set management and acts as an easy to implement supplement to existing systems and processes, rather than adding overhead and cost.

The methods used must be vendor and product independent, and be easily extensible to additional products.

Approach

To meet our goal of producing a minimally intrusive and maximally effective tool, we elected to restrict ourselves to analysis of security configurations (e.g., firewall configurations and router ACLs), and more specifically to the structural properties of the rule sets.

We do not evaluate whether a given configuration enforces the 'correct' policy, or adheres to some specific set of best practices which may or may not be applicable.

These decisions have multiple advantages – analysis takes place offline, with no requirement for agents or special access to the security infrastructure. Analysis is repeatable, and can be used to show improvement (or lack of improvement) in the configuration(s) over time. Further, as the analysis is based on the structural properties of the rule sets, it is vendor agnostic, and allows for rule set comparison between different products.

We first identify the minimal set of information required to describe the action of the rule sets from various configurations, which allows us to translate the configurations to a ''standard'' grammar. As a result, it becomes extremely straightforward to handle and compare multiple types of configurations.

We next identify a set of universal issues that typically cause hard to predict, difficult to diagnose (or inexplicable) effects on firewall management, and then develop a set of common ideal conditions for firewall rule sets to avoid those issues, and define metrics which can be used to measure the degree to which a given firewall rule set differs from the ideal.

Finally we describe the implementation and use of the prototype tool to improve the management of production firewall rule sets, and lessons learned.

Sidebar: Terminology

Location: Source or destination in a rule
Service: Port or protocol
Action: What to do when a rule matches
Rule: A source, destination, service & action combination
Object: Location or Service
Block: A subset of a Rule or Object

For the purposes of this paper, a firewall rule set is a collection of declarations consisting of a source location, destination location, service and action declaration. We do not address transformations such as NAT or PAT in this paper, although the work is extensible. Since each location and service can be a group of locations and services, and there are no constraints preventing overlapping locations or services, it is unfortunately simple to define (and hard to discover) rules which are partially or completely similar to other rules.

Realization

There are policy and vendor independent characteristics that can be generalized as being universally common to a well managed firewall rule set, and which can be used as a basis for metrics to evaluate and improve firewall rule sets.

We posit that the ideal structural properties of a firewall rule set are:

- Non-Interference
  - Rules should not interfere with other rules. Rules should not partially or completely overlap other rules, or be partially or completely overlapped by other rules except in the case where the more specific rule is acted on first, and is completely overlapped by the less specific rule, and the action of the less specific rule differs. Overlapping rules are described as 'eclipsing' or 'eclipsed' rules, and are broken down into 'interfering blocks.'
  - Objects are unique. Objects do not define the same location or service.
- Simplicity
  - There are no unused non-default objects defined
  - Only rules which can be triggered are defined in the rule set
  - Rules permit only what is required by policy
- Consistency
  - Rules actions are consistent. If rules interfere, except in the case noted above, they should have the same action.
  - Object naming style is consistent if named objects are used.

Unfortunately, since firewalls are highly idiosyncratic, it is not possible to discuss firewall rule sets without noting the existence of configuration and device specific corner cases.

- Object Template Re-Use: Objects in multiple firewall policies used by a common management interface must remain identical across all firewalls using them.
- Rule Set Commonality: Rules which can not be triggered may only be permitted if a single rule set common to multiple firewalls is in use.

Interpretation

Given the ideal characteristics shown above, we describe a set of metrics that we can use to measure how well a firewall rule set is managed. In order to be meaningful across multiple firewalls, and multiple types of firewalls, the metrics selected must also be policy independent and vendor independent.

We use Non-Interference and Consistency to provide an understanding of the complexity of a given firewall rule set, while Simplicity and Effectiveness measure the functionality of the rule set.

- Non-Interference (Rules): The fraction of rules in a firewall rule set which do not interfere with other rules.
- Non-Interference (Objects): The fraction of objects in a firewall rule set which do not define the same location or service.
- Simplicity (Rules): The fraction of rules in a firewall rule set which can be triggered.
- Simplicity (Objects): The fraction of objects in a firewall rule set which are defined and used.

Since we have explicitly stated that policy analysis is out of scope for the purposes of our analysis, we will avoid the question of how to measure 'permit only what is required by policy' at this time.

- Consistency (Rules): The fraction of interfering rules in a firewall rule set which have the same action.

Measuring inconsistent object naming style and object template re-use is challenging, and is left to future work.

Based on the above properties, we propose a new metric – *effectiveness* – that can be used to evaluate the complexity of a rule set. This metric essentially captures the degree to which different rules are independent of one another; the intuition is that the greater the overlap, the more complex the rule set and hence more costly to manage. This will also allow us to track the effectiveness of firewalls over time as their rule sets evolve.

- Effectiveness: A measure of the fraction of a given Rule or Object that is not interfered with.

We also investigate other metrics such as frequency of log hits and interference counts, simplicity measures, and consistency measures over rules and objects.

Implementation

Our prototype tool currently handles Checkpoint configurations; the Checkpoint configuration file formats are notably different from the single file, single line style configurations of most other firewalls. We have done proof of concept checks against Cisco PIX/ASA, pf and ipfilter to confirm our ideal state hypothesis, but have not yet implemented parsers for those configurations.

Given two rule sets, the tool produces a comprehensive list of the traffic that one rule set will let through but not the other one. As such, we can use it to compare the existing rule set with a second rule set containing the proposed changes. The administrator can visually check if the difference in patterns of allowed packets corresponds to what he or she intended in proposing the changes.

The tool is implemented in Java SDK 1.6. It can be invoked either from a command line or from a Web interface. This Web interface is implemented using Jetty (Version 6.1). The tool requires the configuration files (object files and rule files) to be transferred to a directory accessible to the firewall analyzer (read only permission is enough). The user can use the web front end to explore the results of the analysis, compare the results of two analyses and run queries via a form interface.

The raw configuration files are parsed using a recursive descent parser that converts the raw configuration files to an intermediate format. The tool interprets rules and objects (represented in this intermediate format) geometrically and computes overlaps between them using computational geometry algorithms. These results are stored in a data structure with multiple indices (rule ids, object ids etc.) so that they are efficiently retrievable by the servlet and query algorithms. (Details in http://www.hpl.hp.com/techreports/2007/HPL-2007-154R1.html .)

Case Studies

The challenges described by firewall managers can be split into three groups – comparison, remediation and reporting.

The anonymized examples below are taken from live production environments, and showcase the use of the tool and metrics which we have described for firewall rule set comparison, remediation and reporting.

Case Study 1: Rule Set Comparison

In this case study, we describe the use of our tool to compare, identify and resolve the differences between two ostensibly identical rule sets.

An end-of-life Checkpoint Firewall-1 NG FP3 firewall was scheduled for replacement with a pair of redundant firewalls running Checkpoint NGX R60, centrally managed by Checkpoint Provider-1.

As the Checkpoint configurations were not compatible between versions, and Checkpoint did not provide a migration tool, a manual rule and object transfer between the old and new environments was required. Since this migration was a bug-for-bug firewall rule set migration, the task of manually re-entering the firewall rules from the old environment to the new environment was delegated to front line support staff, with validation of the copied rules being performed by a senior engineer.

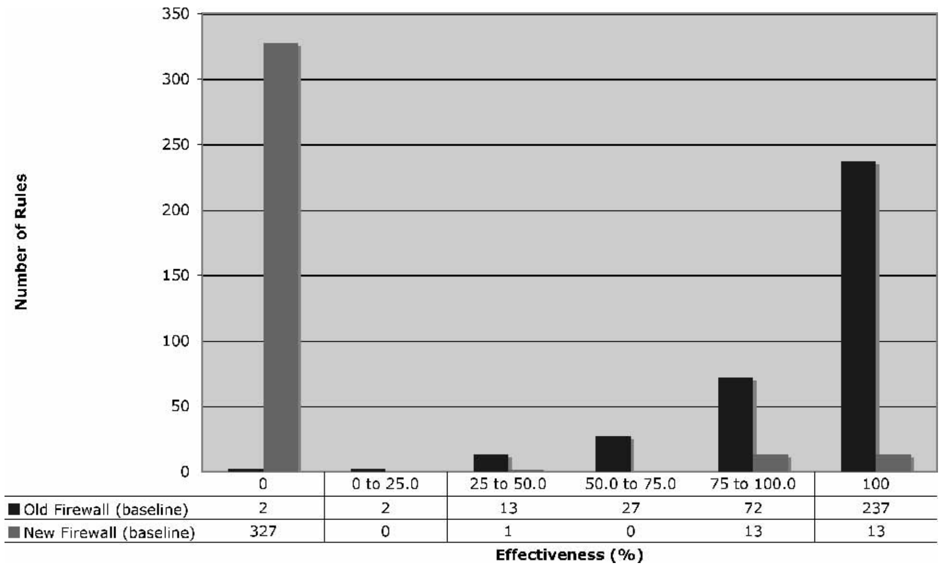
An initial comparison of the old and new rule set rule effectiveness made it immediately obvious that the two configurations were significantly different.

The gross difference in rule sets was swiftly explained by noting that although the total number of active rules was correct (the new firewall has an additional rule for state synchronization), the old configuration had six deny rules, while the new configuration had 12.

A quick visual comparison of the two rule sets also revealed a number of rules that were missing, out of order, with incorrect actions, or missing objects.

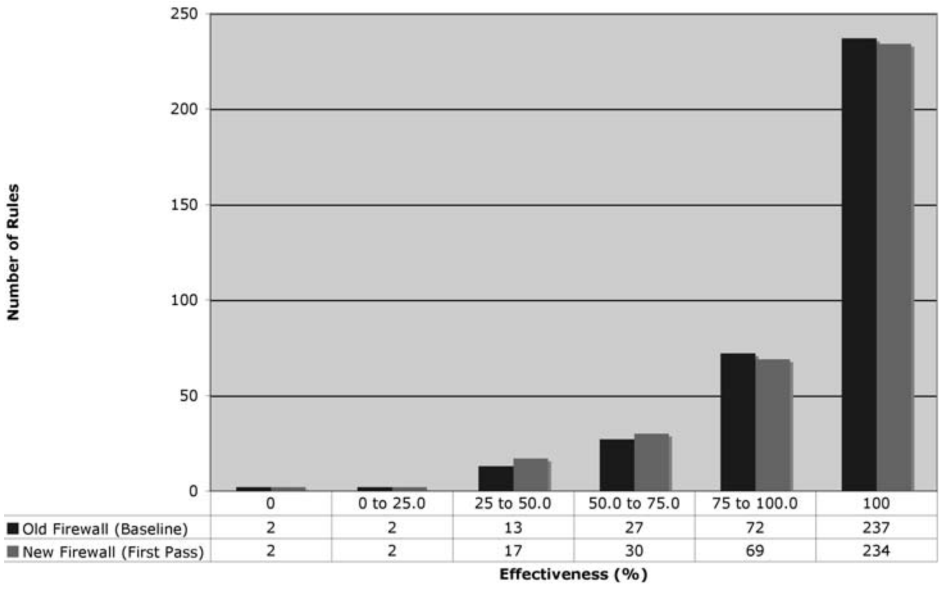
Source	Destination Firewall	New Firewall Action	New Firewall Rule ID	Old Firewall Action	Old Firewall Rule ID
10.0.6.230	192.168.10.21	322	accept	352	drop
10.0.6.231	192.168.10.21	322	accept	352	drop
10.0.6.232	192.168.10.21	322	accept	352	drop
172.16.0.0-172.16.32.20	192.168.11.150	311	accept	352	drop
172.16.0.0-172.16.32.20	192.168.10.150	353	drop	310	accept
172.16.32.22-172.16.35.255	192.168.11.150	311	accept	352	drop
172.16.32.22-172.16.35.255	192.168.10.150	353	drop	310	accept
172.16.64.0-172.16.72.255	192.168.11.150	311	accept	352	drop
172.16.64.0-172.16.72.255	192.168.10.150	353	drop	310	accept
172.16.128.0-172.16.144.255	192.168.11.150	311	accept	352	drop
172.16.128.0-172.16.144.255	192.168.10.150	353	drop	310	accept

Table 1: Case Study 1 – Rule set comparison – Object TCP-3389 interference.



	0	0 to 25.0	25 to 50.0	50.0 to 75.0	75 to 100.0	100
■ Old Firewall (baseline)	2	2	13	27	72	237
▨ New Firewall (baseline)	327	0	1	0	13	13

Figure 1: Case Study 1 – Rule set comparison baseline.



	0	0 to 25.0	25 to 50.0	50.0 to 75.0	75 to 100.0	100
■ Old Firewall (Baseline)	2	2	13	27	72	237
▨ New Firewall (First Pass)	2	2	17	30	69	234

Figure 2: Case Study 1 – Rule set comparison first pass.

These issues were straightforward to identify, and correct, and would certainly have been identified through manual examination. Once the first pass for gross errors was complete, the effectiveness of both the old and new rule sets was nearly identical.

	Old Firewall (Baseline)	New Firewall (Baseline)
Drop Rules	6	12
Accept Rules	347	342
Total Rules	353	354

Although the configurations appear to be nearly identical from a rule effectiveness standpoint, comparing used objects showed a high number of objects in use in only one of the two configurations – 65 objects unique to the old configuration, and 77 objects unique to the new configuration. 100 objects in the two configurations interfered.

Further, as demonstrated by the object interference example in Table 1, each object could interfere with multiple rules, and either partially or completely.

More specific examination of the interfering objects revealed that the object definitions also varied between the old and new firewalls, and object names had not been consistently defined between the old and new firewalls. Object names had been entered with varying case (TCP vs tcp vs Tcp), different separating characters (TCP-22 vs TCP_22 vs tcp22) and with completely different names (TCP-22 vs SSH).

Resolving the interfering objects then became an iterative process of resolving one set of interference and using the tool to compare the configurations again, as resolving interference in one rule frequently affected interference in other rules.

Case Study 2: Rule Set Remediation

In this case study, we describe the use of our tool in combination with log file analysis to identify partially and completely ineffective rules, and then examine the effect of removing the identified rules from the rule set.

We were asked to identify what rules in a given set of approximately 350 rules were not being used, based on six months of firewall logs, and identify the impact(s) of removing those rules.

The method we selected was a combination of configuration analysis and log analysis. Log analysis was used to identify those rules which had few or no hits during the monitored period, and could be removed, while configuration analysis was used to identify eclipsed rules, and the impact of removing rules.

As shown in Figure 3, log analysis revealed that a sizeable number of rules had received no hits at all during the six month analysis period.

Based on input from the customer and review of the rules, the decision was made to remove all rules which had fewer than 1,000 hits over six months, with the exception of two specified IP ranges.

Based on that specification, a new version of the rule set was created, and the old and new rule sets compared to determine the impact of the proposed changes.

As shown in the figures below, removing the seldom and never used rules resulted in a dramatic reduction in both the number of rules and the number of objects in use.

While the number of rules and objects in the new configurations had decreased significantly, there were still a number of interfering objects and eclipsing rules

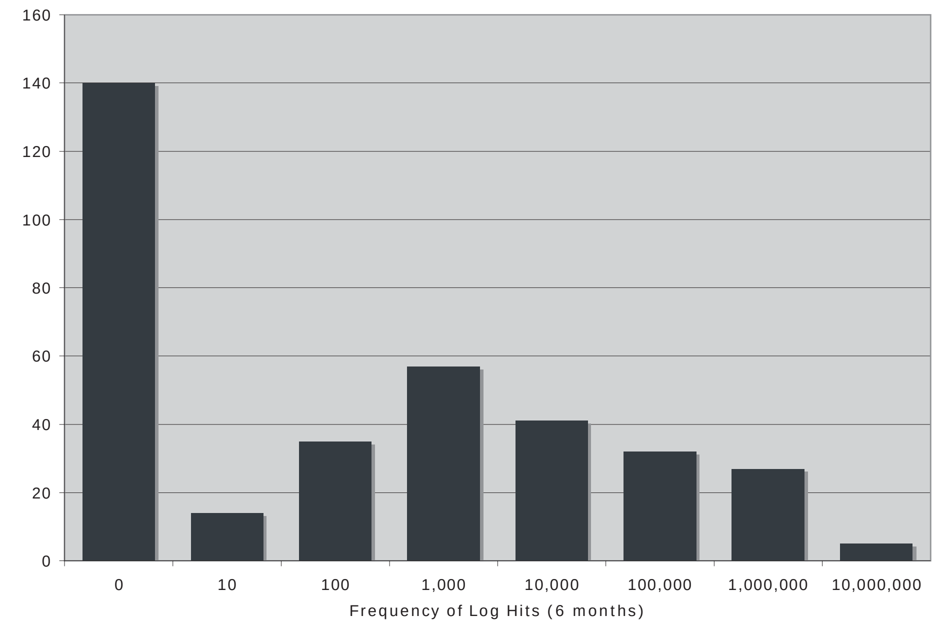


Figure 3: Case Study 2 – Rule set remediation – Rule hit frequency figure.

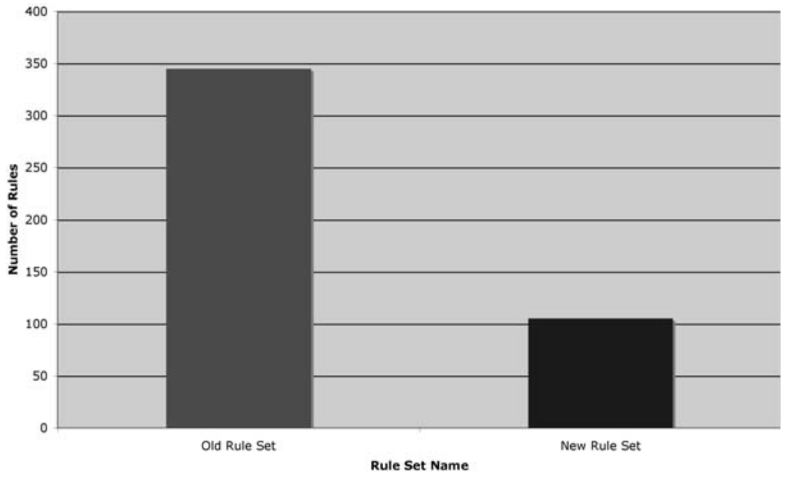


Figure 4a: Case Study 2 – Rule remediation – Number of rules.

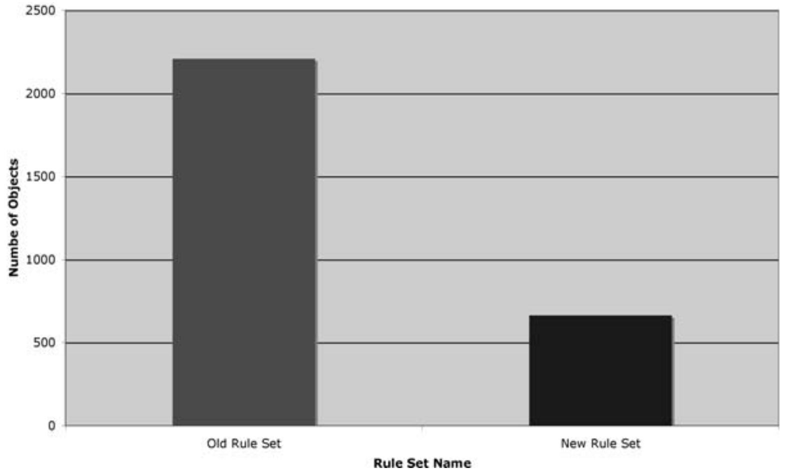


Figure 4b: Case Study 2 – Rule remediation – Number of objects.

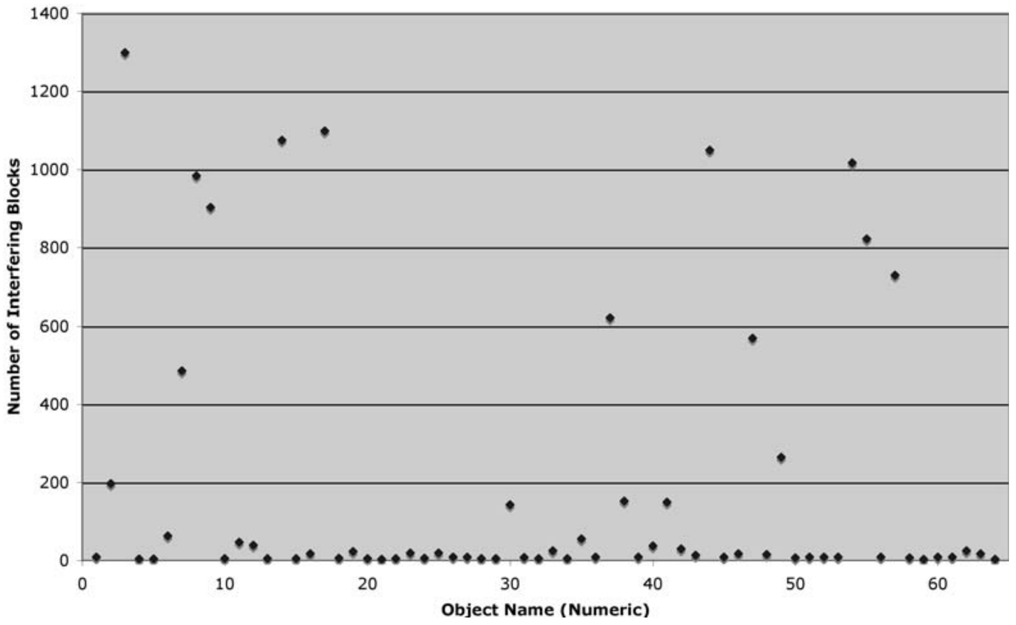


Figure 5: Case Study 2 – Rule set remediation – Number of interfering blocks in rule set.

in the new configuration, pointing towards a number of additional rule set improvements which we did not investigate at that time.

Rule Set Reporting

We previously described six metrics which we believe are a good measure of how well a firewall rule set is managed. In the first four sections we apply these metrics to a collection of 50+ Checkpoint Firewall-1 configurations spanning 2+ years, 34 different firewalls and multiple business types, and discuss the results; the last section looks at a single firewall over time, and through a migration to new hardware.

- Non-Interference (Rules): The fraction of rules in a firewall rule set which do not interfere with other rules.
- Non-Interference (Objects): The fraction of objects in a firewall rule set which do not define the same location or service.
- Simplicity (Rules): The fraction of rules in a firewall rule set which can be triggered.
- Simplicity (Objects): The fraction of objects in a firewall rule set which are defined and used.
- Consistency (Rules): The fraction of interfering rules in a firewall rule set which have the same action.
- Effectiveness: A measure of the fraction of a given Rule or Object that is not interfered with.

Non-Interference

As one might hope, in general, as the number of rules in a firewall rule set increases, the number of

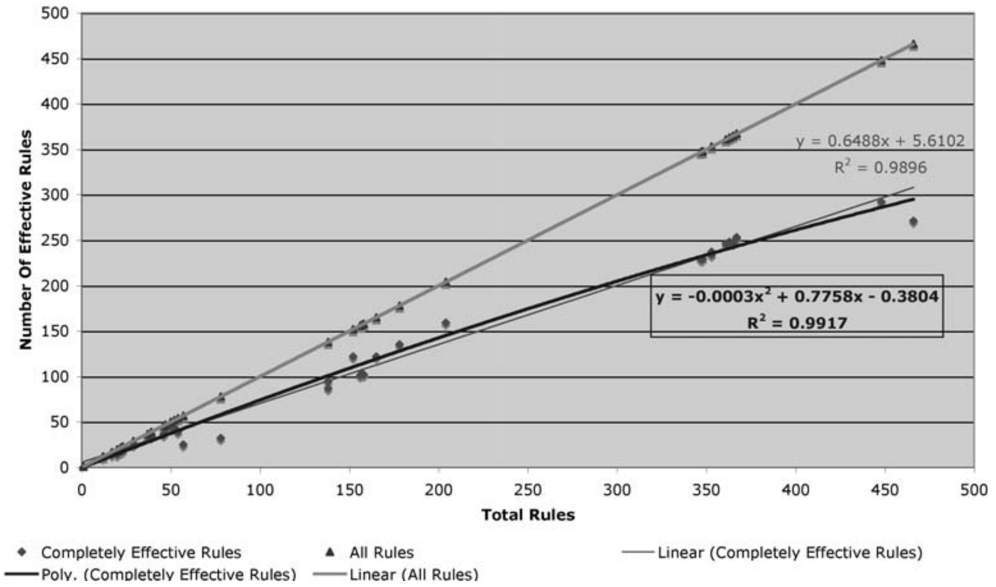


Figure 6a: Case Study 3 – Rule set reporting – Non-interference (rules).

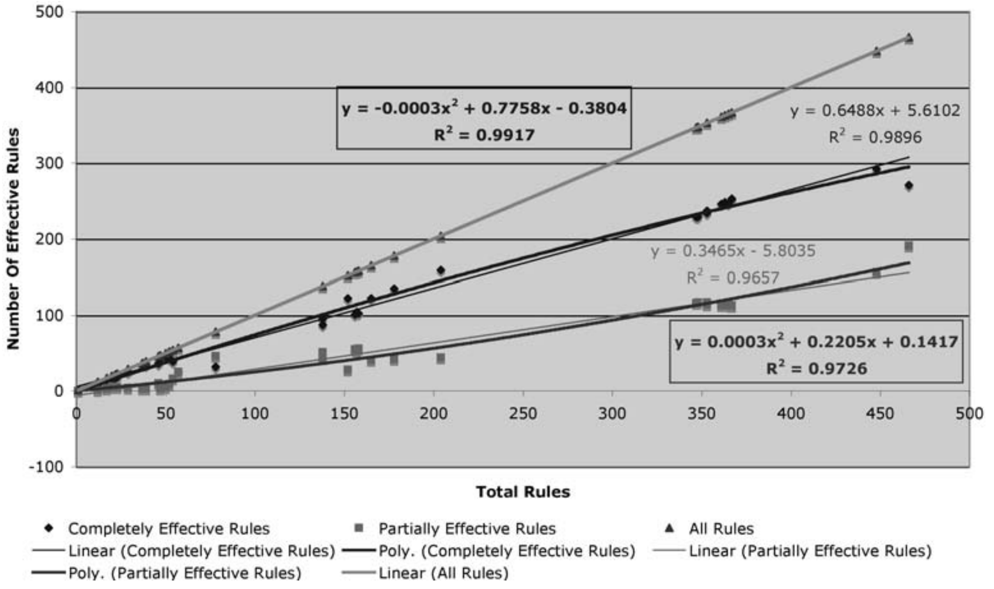


Figure 6b: Case Study 3 – Rule set reporting – Non-interference (rules).

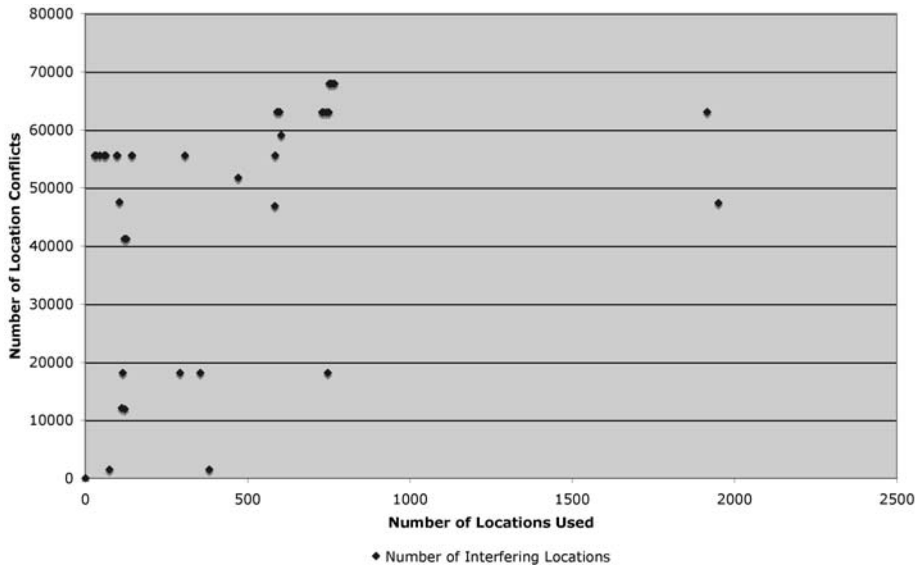


Figure 7: Case Study 3 – Rule set reporting – Non-interference (objects).

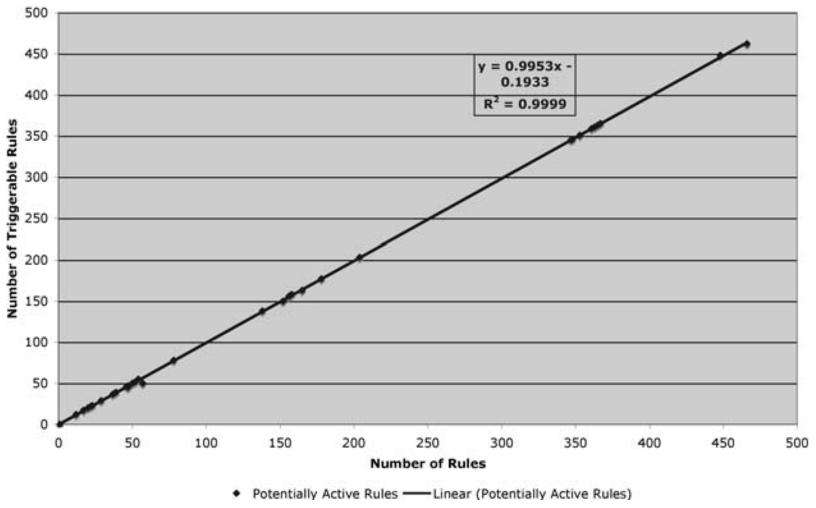


Figure 8a: Case study 3 – Rule set reporting – Simplicity (rules).

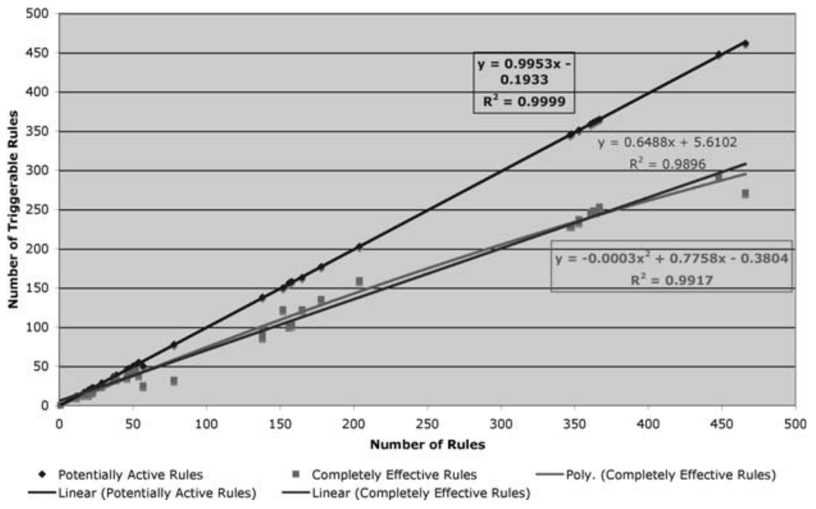


Figure 8b: Case study 3 – Rule set reporting – Simplicity (rules).

non-interfering rules also increases. However, as shown in Figure 6a, as the number of rules increases, the number of non-interfering rules drops away from the total number of rules.

More clearly, Figure 6b suggests that the number of interfering rules is proportionate to, and increases with, the total number of rules. There is also an interesting hint that the number of non-interfering and interfering rules may intersect, and that the number of interfering rules will exceed, and eventually overwhelm the number of non- interfering rules, for a sufficiently large number of rules.

As Checkpoint uses shared object definitions for all firewalls being managed by the same management station, the number of non-interfering objects is heavily dependent on the number of objects defined in that management instance, as the loose plot below demonstrates. It is clear, however, that interference between objects is to be expected in all rule sets, and increases with the number of objects defined on a given common management instance.

Simplicity

If we consider Simplicity of rules strictly from the standpoint of rules which could be triggered – that is to say, any rule which is not completely eclipsed, Figure 8a shows that the relationship between number of rules, and number of potentially active rules is linear, and almost exact.

This is misleading, as Figure 8b shows; as the number of rules increases, the number of completely effective rules immediately drops away from the number of potentially active rules, showing that rules can be expected to interact in unexpected ways. The data obtained for object simplicity is inconclusive, and suggests that our approach for examining objects in Checkpoint configurations managed by a shared management instance needs to be revisited.

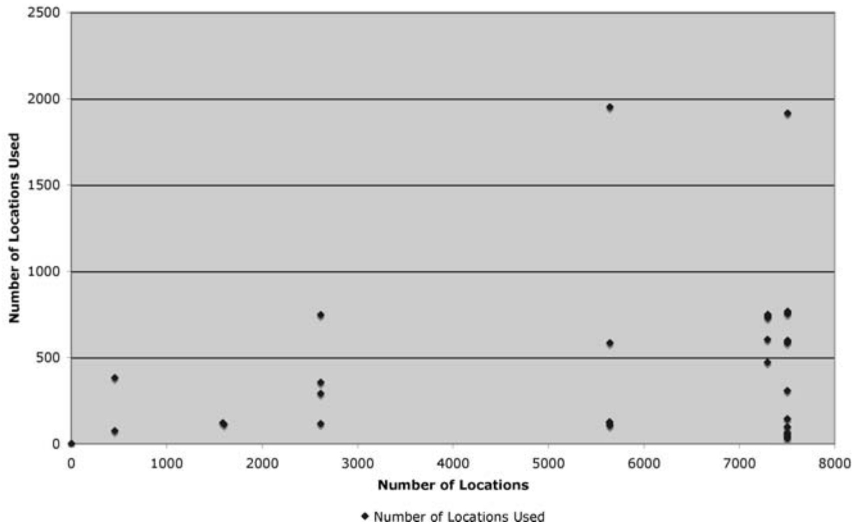


Figure 9: Case Study 3 – Rule set reporting – Simplicity (Objects).

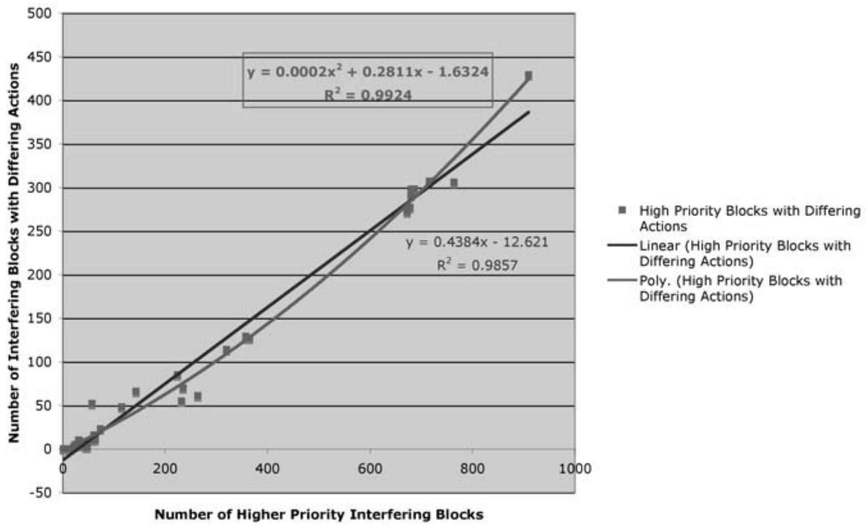


Figure 10: Case Study 3 – Rule set reporting – Consistency.

Consistency

Here we look specifically at the case of interfering rules, where the action of the rule taking priority differs from the action of the rule(s) being eclipsed.

Unsurprisingly, we see that the number of interfering rules with inconsistent actions increases as the size of a rule set increases. We have not calculated the case where a more specific rule is acted on first, and is completely overlapped by a less specific rule separately, but other configuration analysis work suggests that this case does not account for the majority of rule interference.

Rule Effectiveness (%)	R^2 Value (poly)
Ineffective Rules	0.6684
0-25	0.6367
25-50	0.9195
50-75	0.9429
75-100	0.9449
Partially Effective	0.9726
Completely Effective	0.9917

Table 2: Case Study 3 – Rule set reporting – Effectiveness.

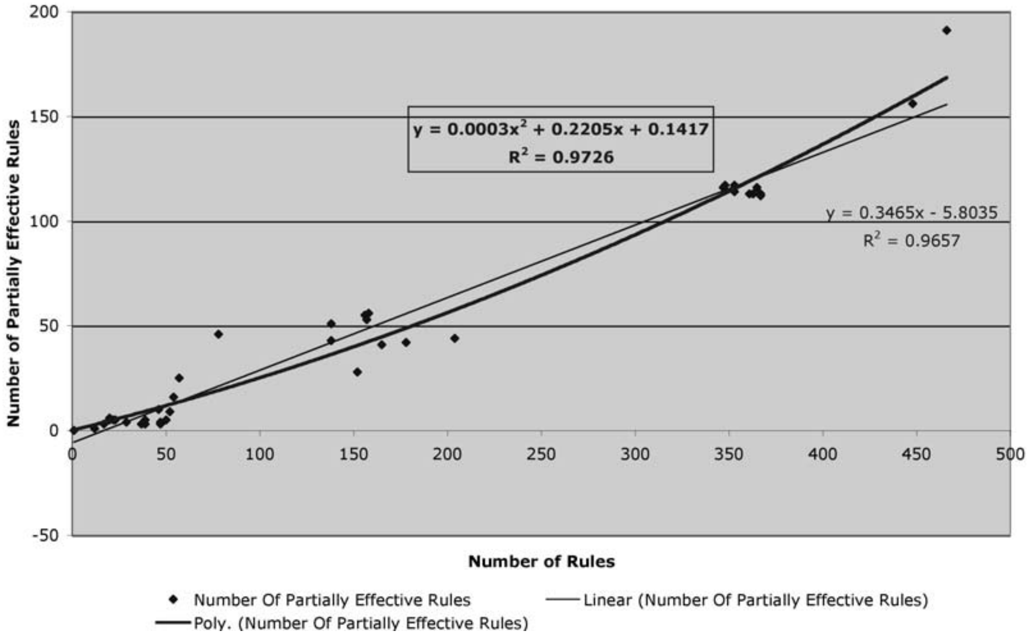


Figure 11a: Case Study 3: Rule set reporting – Effectiveness.

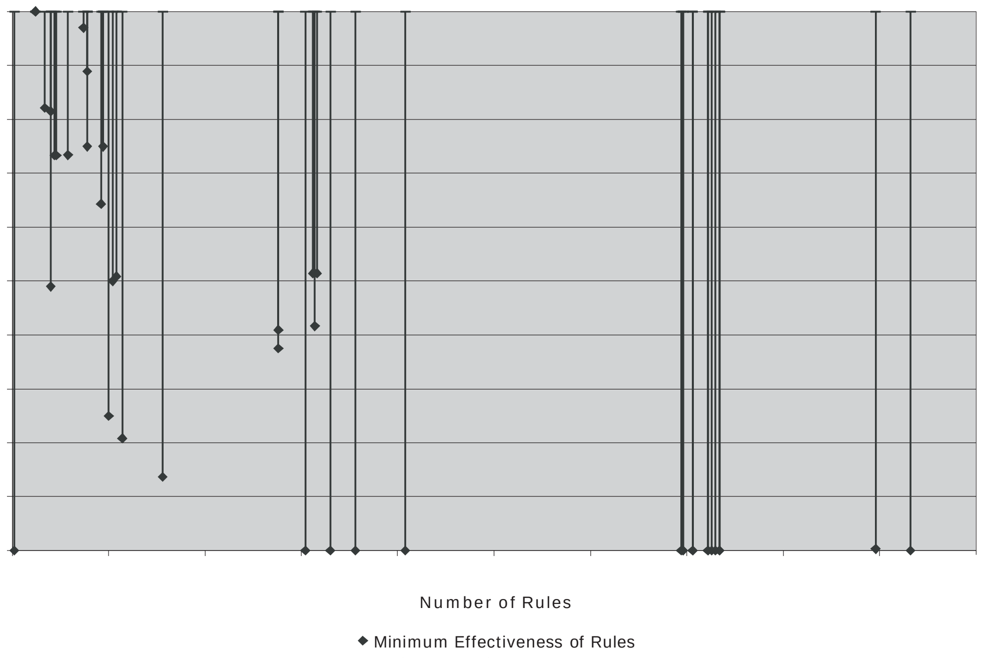


Figure 11b: Case Study 3: Rule set reporting – Effectiveness.

Effectiveness

As shown above, there is a direct correlation between the number of rules, and the number of partially effective rules. Less obviously, the variance in rule effectiveness also increases as the rule set size increases. While the overall R^2 value for partially effective rules is excellent, breaking the rule effectiveness into ranges is suggestive. The lowest correlation values are associated with the most ineffective rules, suggesting that these rules may be easier to detect and resolve.

Trends Over Time

The following data is drawn from an ongoing firewall migration project which has been in progress for nearly two years, and continues onward with an extended period of overlapping operation for the old and new firewalls.

When the configuration was initially migrated from the old firewall to the new firewall, a manual cleanup of the rule set took place; currently most rule set changes are expected to be implemented on both the old and new firewalls.

Unsurprisingly, we see that both the old and new rule sets show a steady increase in the number of rules and locations defined over time

Similarly, we also see the number of partially effective rules increasing over time, as the number of rules in the rule set increases.

The number of partially effective rules in the new firewall rule set is initially lower than the number of partially effective rules in the old firewall rule set, thanks to a manual clean up of the rule set, prior to migration. It is abundantly clear, however, that the effect of the rule set clean up was only temporary.

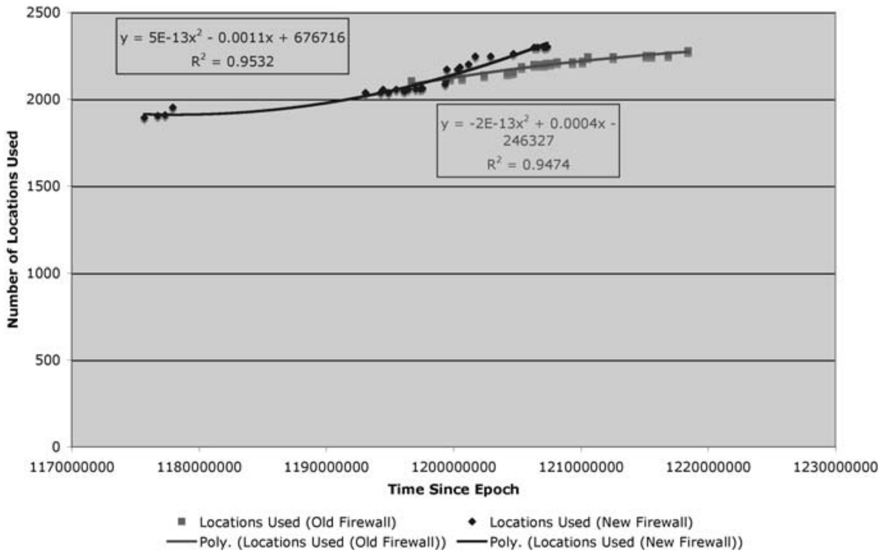


Figure 12a: Number of locations used over time.

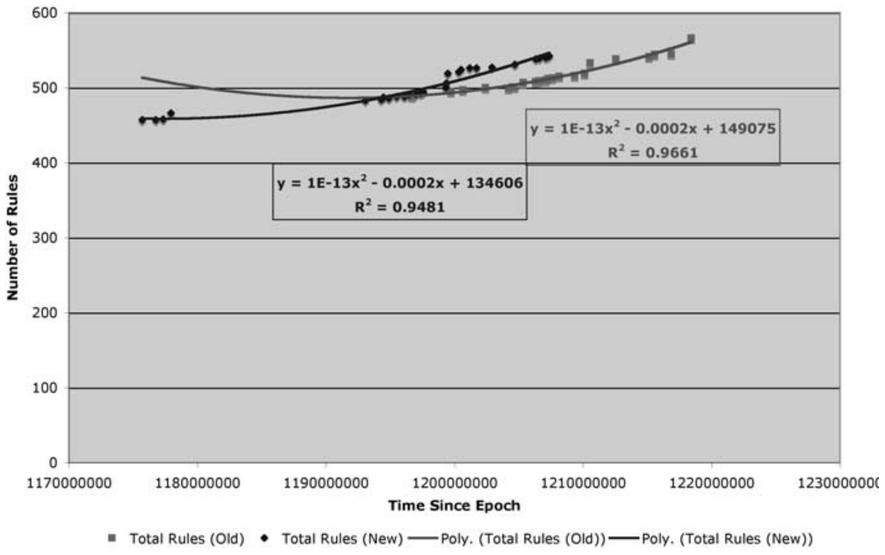


Figure 12b: Number of rules over time.

Further, if we examine Figure 13b, it is clear that the majority of rule changes which result in partially effective rules are for rules in the 75-100% effectiveness range.

Discoveries

Our findings in this work range from confirming relatively obvious intuitions (such as simpler configurations are better) to some more surprising results. Our discoveries here summarize the case studies, as well as additional experiences not described in this paper.

1. The number of rules in a rule set is highly correlated to the number of partially and completely effective rules. This suggests that our intuition that larger rule sets are more complex is likely to be correct.
2. There is a high correlation between the number of interfering rules, and the number of interfering rules with conflicting actions. While it is likely that some of these interfering rules have been knowingly configured (e.g., a permit rule for a host in a larger subnet which is denied), in practice it appears that most interfering rules are unintended.
3. The majority of partially effective rules are 25-99% effective. Further, when a rule set undergoes a manual clean up process, the number of completely ineffective rules is typically reduced dramatically, while the number of partially effective rules remains relatively unchanged. This suggests that complex interference patterns are more difficult for people to detect and resolve.
4. The effectiveness of a rule set decays visibly over time. If a rule set is cleaned up, the effectiveness immediately begins to decay again.
5. Most firewall rule sets have some amount of interference, but the amount of interference

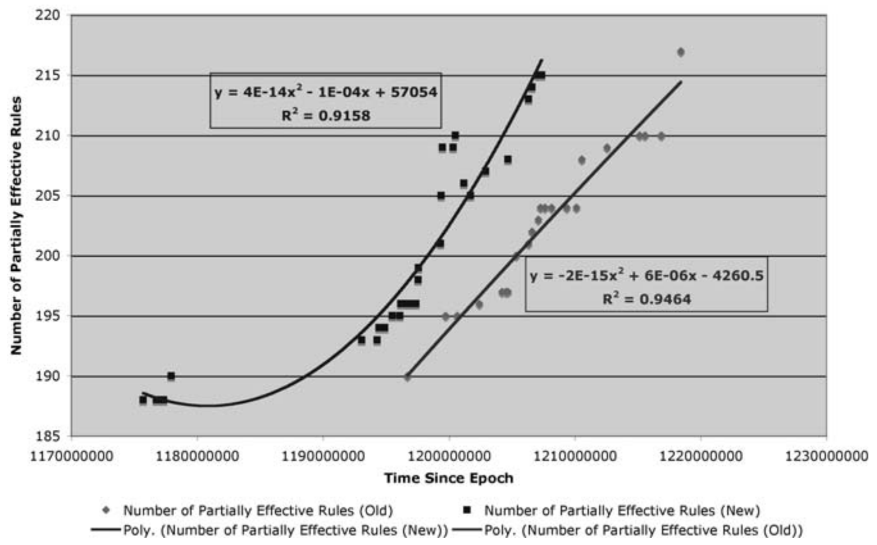


Figure 13a: Trends over time – Rule interference.

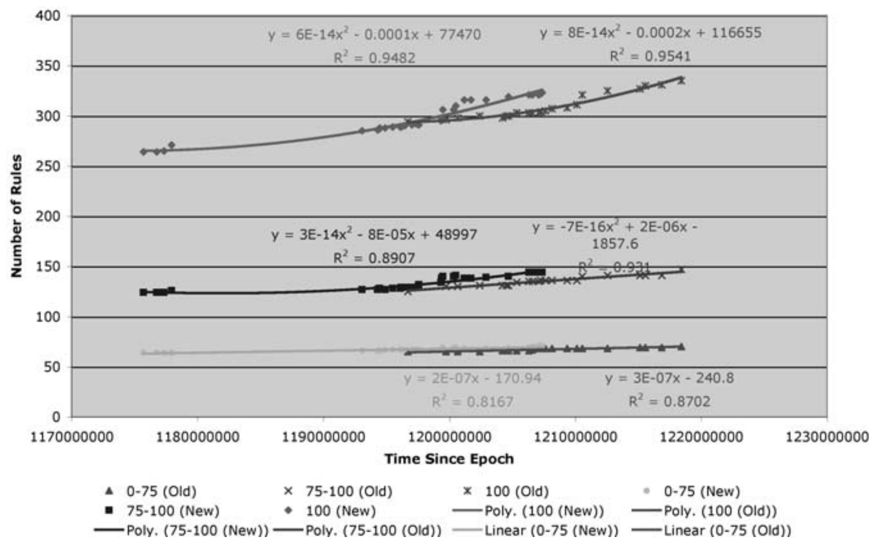


Figure 13b: Trends over time – Distribution of paritally effective rules.

varies dramatically, and increases as number of rules increases.

6. While effectiveness varies dramatically, in general, more effective rule sets have less variance in effectiveness, suggesting that less confusing rule sets are easier to manage.
7. In general, in the absence of a major clean up effort, the number of rules and objects used in a given rule set always increases. Since the number of partially and completely effective rules also increases as the number of rules increase, this points to the management of firewall rule sets as write-once, remove-never devices.
8. Even when a rule set is cleaned up, the number of rules decreases, but the number of objects remains constant or increases, suggesting that object management is an ongoing challenge.
9. There is no clear relationship between the number of rules and number of locations.

Unfortunately, it appears that objects are a weak source for information about the state of rule sets, at least as we are currently measuring and examining objects. It is likely that the use of shared objects by all firewalls managed by a given Checkpoint management instance is the cause of this issue, and we hope to better address the role of objects as a means of measuring firewall rule sets in the future.

Ultimately, our holy grail is to be able to correlate the above metrics with quantities dear to higher level management such as cost, 'ease of management' etc. We need to measure these 'manager-friendly' metrics to see what configuration and network metrics correlate with them – this is a part of our ongoing efforts.

Usage

The tool and metrics we describe are extremely useful for rule set comparison and clean up, as they automate the process of finding conflicts, vastly reducing the amount of time and effort required.

As well as being used for massive rule set remediation projects, we suggest that an ideal usage would be to include rule set analysis as a part of routine change management, to identify unanticipated effects from rule and objects additions and removals.

We also hope that our metrics work will serve as a measure of health for firewalls – and can drive decisions like "when should I clean up my rule set?" or "Is my outsourced firewall administrator doing their job properly?"

Future Work

Going forward, we intend to improve our understanding of firewall management. Specific areas of interest include:

- Analyzing configurations from a wider variety of firewalls to discover commonalities and consider additional metrics.
- Tracking the effect of using the metrics we have described to improve firewall management over time.
- Correlating the metrics we have described to the rate of change, number of incidents, and time to resolve incidents in the environment.
- Correlating the metrics we have described to cost.

Related Work

There has been a flurry of work on firewall analysis in recent years. The most relevant is the Firewall Analyzer (FA) product from Algorithmic Security Inc [ALG] that builds on previous research reported in [MWZ] and [WOO]. FA analyzes a number of different types of firewalls, reports common vulnerabilities found in the rule set, and detects rules that are redundant (i.e., are eclipsed by higher-priority rules). FA does not check equivalence of rule sets.

The Solsoft Firewall Manager [SOL] converts high-level specifications of allowed and prohibited traffic into firewall rule sets. However, it does not analyze existing rule sets on a firewall to see if they comply with the high-level specifications.

There are also a number of academic papers on firewall analysis. The paper [AH1] recognizes different types of conflicts that can occur between a pair of rules, but does not address the more general problem of one rule being eclipsed by a set of rules. The authors extend their study to rules on multiple firewalls in a tree network [AHB], but the results are again limited to interactions between pairs of rules. The paper [GLI] proposes a simplistic way to design rule sets such that every packet is associated with exactly one rule; the problem with this approach is that one can design much more compact rule sets if multiple rules (resolved by a priority mechanism) can apply to any packet. The paper [EMU] investigates efficient data structures to process basic firewall queries.

The paper [VPR] addresses problems of generating and analyzing rule sets for networks with multiple firewalls and addresses the problem of checking equivalence of rule sets. However, the paper relies on exhaustive analysis of all possible packets, and it is unclear whether the methods scale to large networks. The paper [BRR] describes a tool to configure and analyze rule sets for networks with multiple firewalls. It employs efficient algorithms that scale for large networks, and while the techniques are relevant for checking rule set equivalence, the paper does not explicitly address the equivalence problem.

The paper [MKE] proposes the use of binary decision diagrams to analyze rule sets. These diagrams allow you to query the firewall rule set but do not easily support the comparison of two rule sets.

Conclusions

Offline configuration analysis is a fast, lightweight, and reliable way to identify the effectiveness of existing firewall configurations, improve maintainability, hasten debugging, and assist with policy, audit and compliance.

Author Biographies

Sandeep Bhatt is a researcher in the Systems Security Laboratory at HP Labs. Since joining HP Labs in 2004, he has focused on the management of distributed access control in enterprise applications, on algorithms for firewall analysis, and on end-to-end access control in enterprise networks. Before joining HP Labs, Sandeep led the Systems Performance team at Akamai Technologies, and was Director of Network Algorithms at Telcordia Technologies where he led the development of fault diagnosis systems for large-scale telecommunication networks. Sandeep was also on the faculty of Computer Science at Yale University, with appointments at Caltech and Rutgers University. His research interests have included algorithms for parallel computation, network communication, and VLSI layout. He received Ph.D, S.M and S.B degrees in Computer Science from MIT.

Cat Okita has more than 10 years of experience as a senior systems, security and network professional in the Financial, Internet, Manufacturing and Telecom sectors. She has designed, managed and contributed to a variety of geographically diverse projects and installations. Her assignments have included proposing and seeing to completion a wide variety of security, Internet, enterprise and monitoring projects, including highly available, redundant fault tolerant systems for security, web services, logging, monitoring, statistics gathering and analysis. Her research interests include privacy, reputation and practical security.

Cat has spoken at LISA and Defcon about identity and reputation, co-chairs the 'Managing Sysadmins' workshop at LISA, and programs for fun, in her spare time.

Prasad Rao is a member of the Systems Security Laboratories within HP labs, where he works in security analytics and metrics. At HP he has been active in the Vantage project that analyzes end-to-end access control across various layers in the enterprise. As a part of this project, he has built technologies to analyze firewalls with very large rule sets (more than 7500 rules) and objects (more than 12000 objects) for presence of operator errors and vulnerabilities – this technology is currently being tested within HP. In addition he has contributed to ongoing work in other areas such as role discovery, securing printing software and privacy compliance.

At Telcordia Technologies he was the technical lead and architect of the Smart Firewalls project (a part of the Dynamic Coalitions Program). Additionally, at Telcordia technologies he built a deductive rule-based system in large scale worker scheduling programs, which had to optimize worker utilization and cost under arbitrarily stated union rules and practical constraints.

For his Ph.D. thesis in logic programming and deductive databases at the State University of New York at Stony Brook, Dr Rao built designed and implemented the tabling engine of the XSB logic programming language – which set speed records for standard benchmarks in the field of deductive databases compared to the state of the art in the early '90s.

Bibliography

[ALG] Algorithmic Security Inc., *Firewall Analyzer: Make Your Firewall Really Safe*, white paper, 2006, http://www.algosec.com .

[AH1] Al-Shaer, E. and H. Hamed, "Firewall Policy Advisor for Anomaly Discovery and Rule Editing," *Proceedings IEEE/IFIP Integrated Management Conference*, 2003.

[AHB] Al-Shaer, E., H. Hamed, R. Boutaba, and M. Hasan, "Conflict Classification and Analysis of Distributed Firewall Policies," *JSAC*, 2005.

[BRR] Bhatt, S., S. Rajagopalan, P. Rao, "Automatic Management of Network Security Policy," *Proceedings MILCOM*, 2003.

[EMU] Eppstein, D. and S. Muthukrishnan, "Internet Packet Filter Management and Rectangle Geometry," *Proceedings ACM SODA*, 2001.

[GLI] Gouda, M. and X-Y. Liu, "Firewall Design: Consistency, Completeness and Compactness," *Proceedings IEEE International Conference on Distributed Computing Systems*, 2004.

[MKE] Marmorstein, R. and P. Kearns, "An Open Source Solution for Testing NAT'd and Nested iptables Firewalls," *Proceedings LISA '05*, 2005.

[MWZ] Mayer, A., A. Wool and E. Ziskind, "Fang: A Firewall Analysis Engine," *Proceedings IEEE Symposium on Security and Privacy*, 2000.

[SOL] Solsoft Inc., http://www.solsoft.com .

[VPR] Verma, P. and A. Prakash, "FACE: A Firewall Analysis and Configuration Engine," *IEEE Symposium on Applications and the Internet*, 2005.

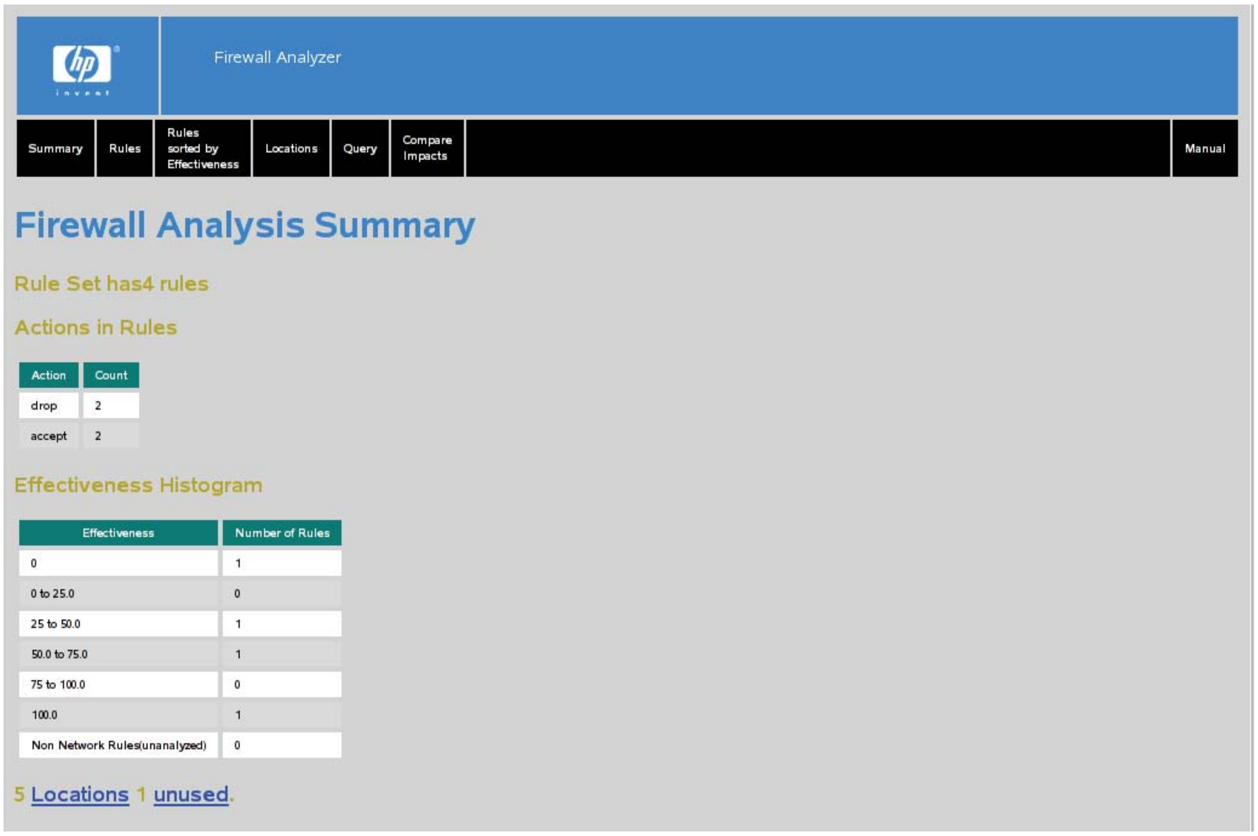
[WOO] Wool, A., "Architecting the Lumeta Firewall Analyzer," *Proceedings 10th USENIX Security Symposium*, 2001.

Appendix 1

To briefly demonstrate the tool, consider the following first match firewall rule set, which contains several errors:

Action	Service	Source	Destination
Permit	SSH	10.0.0.0/8	10.0.0.0/8
Deny	SSH	10.0.0.0/8	10.3.1.0/24
Permit	SSH,https	10.0.0.0/8	10.3.1.61/32
Deny	*ANY*	*ANY*	*ANY*

The tool produces the following overall summary:



Firewall Analysis Summary

Rule Set has4 rules

Actions in Rules

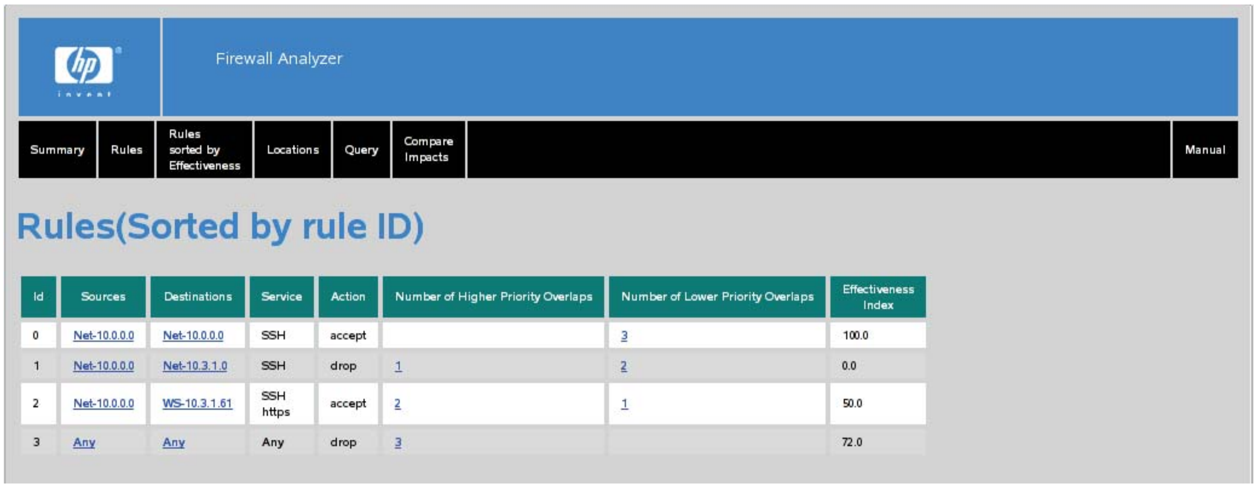
Action	Count
drop	2
accept	2

Effectiveness Histogram

Effectiveness	Number of Rules
0	1
0 to 25.0	0
25 to 50.0	1
50.0 to 75.0	1
75 to 100.0	0
100.0	1
Non Network Rules(unanalyzed)	0

5 Locations 1 unused.

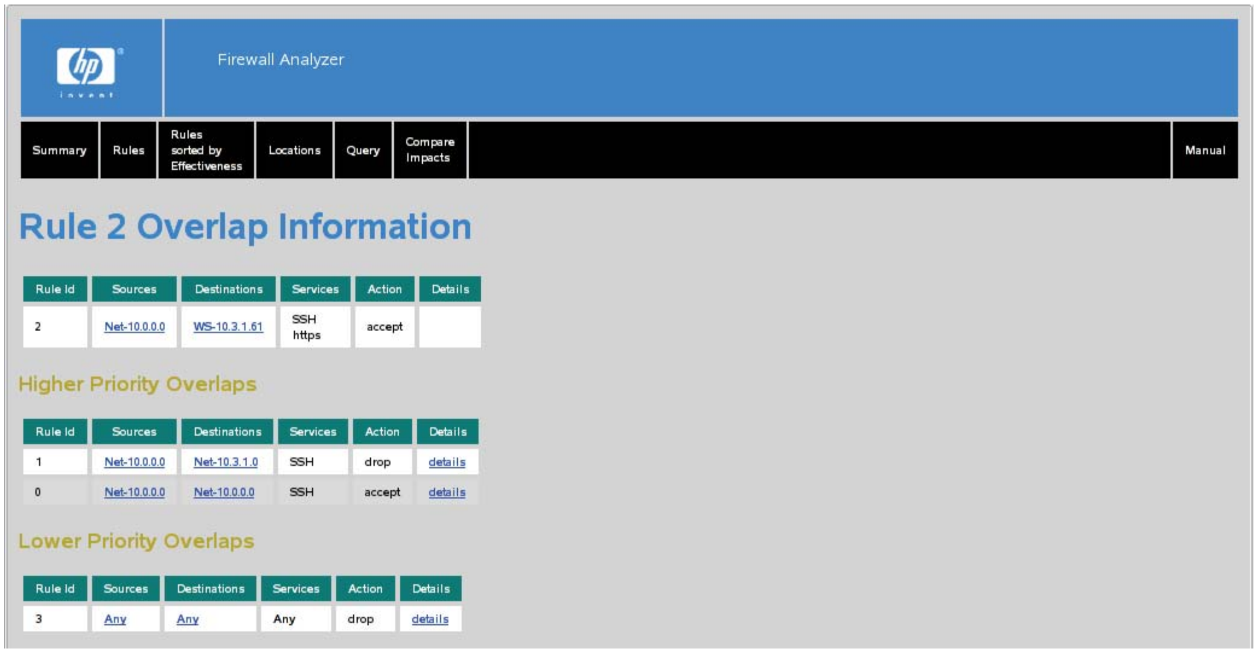
From the Summary, we can then obtain more specific information about the rule set...



Rules(Sorted by rule ID)

Id	Sources	Destinations	Service	Action	Number of Higher Priority Overlaps	Number of Lower Priority Overlaps	Effectiveness Index
0	Net-10.0.0.0	Net-10.0.0.0	SSH	accept		3	100.0
1	Net-10.0.0.0	Net-10.3.1.0	SSH	drop	1	2	0.0
2	Net-10.0.0.0	WS-10.3.1.61	SSH https	accept	2	1	50.0
3	Any	Any	Any	drop	3		72.0

... and we can drill down for detail about the overlapping rules:



Rule 2 Overlap Information

Rule Id	Sources	Destinations	Services	Action	Details
2	Net-10.0.0.0	WS-10.3.1.61	SSH https	accept	

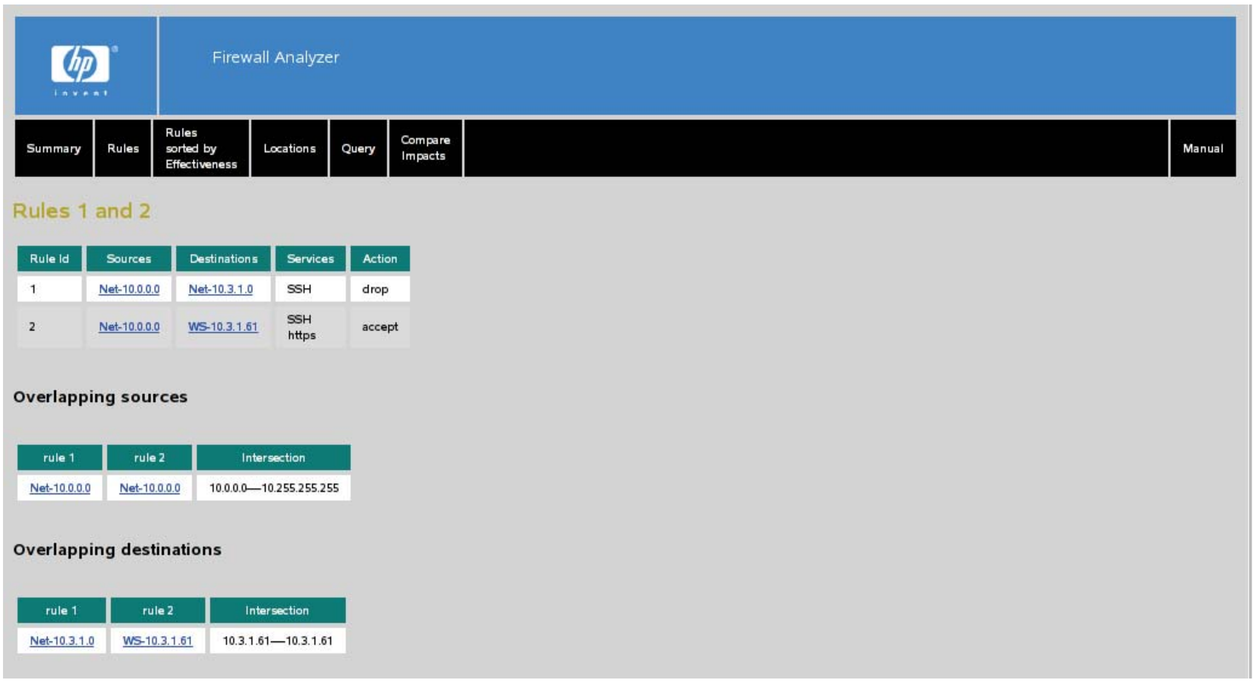
Higher Priority Overlaps

Rule Id	Sources	Destinations	Services	Action	Details
1	Net-10.0.0.0	Net-10.3.1.0	SSH	drop	details
0	Net-10.0.0.0	Net-10.0.0.0	SSH	accept	details

Lower Priority Overlaps

Rule Id	Sources	Destinations	Services	Action	Details
3	Any	Any	Any	drop	details

... and finally procure further details about the specific overlaps:



Rules 1 and 2

Rule Id	Sources	Destinations	Services	Action
1	Net-10.0.0.0	Net-10.3.1.0	SSH	drop
2	Net-10.0.0.0	WS-10.3.1.61	SSH https	accept

Overlapping sources

rule 1	rule 2	Intersection
Net-10.0.0.0	Net-10.0.0.0	10.0.0.0—10.255.255.255

Overlapping destinations

rule 1	rule 2	Intersection
Net-10.3.1.0	WS-10.3.1.61	10.3.1.61—10.3.1.61

Concord: A Secure Mobile Data Authorization Framework for Regulatory Compliance

Gautam Singaraju and Brent Hoon Kang – University of North Carolina at Charlotte

ABSTRACT

With the increasing adoption of mobile computing devices that carry confidential data, organizations need to secure data in an ever-changing environment. Critical organizational data should be protected from a) a disgruntled user's access and b) a theft or loss of the mobile device. When such compromises do occur, future data access should be immediately revoked and the knowledge of the data that might have been exposed be identified. Such assessment enables an organization to demonstrate its adherence to mandated regulatory compliance.

We propose Concord: a framework that provides an organizational service that allows an organization to monitor data that has been accessed on its users' mobile devices. Concord distributes trust among multiple entities so as to enable data access following their successful interaction. Firstly, to enable data access, users of the mobile device require the organization's involvement to access the data on the mobile devices. Likewise, in the event of loss or theft of a mobile device, organizations can immediately discontinue further requests for data accesses to the previously-unread data on the mobile device. Secondly, a valid user's consent is required to access the data. Thus, should an intruder somehow receive organizational permission, the data on the mobile device is still inaccessible. Thirdly, upon identification of a compromise, Concord provides the organization with the detailed information about the data that has been exposed enabling them to initiate steps for regulatory compliance.

Introduction

Today, organizations are faced with numerous cyber threats from internal as well as external sources. Hackers operating from external institutions constantly lurk in search of an IT vulnerability that would allow them to compromise non-public data. Meanwhile, insiders or disgruntled employees have access to confidential information [8]. Notably, the recent increase in the use of mobile devices [11] has increased organizations' risk of losing sensitive data as mobile devices are prone to theft or loss.

Incidents of data leak, theft, or misuse are not rare; regulation compliances [7, 13] have been imposed to guide an institution's data management and security policies. Regulatory compliances are designed to protect the privacy of non-public information. The compliance standards require institutions to monitor each user's data accesses. For example, data breach notification bill states that in case of data loss, institutions should inform the affected customers.

Protecting data is challenging, especially when the data is on mobile devices. We stipulate that a system that protects data should:

a) Safeguard the confidentiality of the data;
b) Enforce immediate access revocation; and;
c) Record the history of data accesses.

To address these requirements, we propose Concord, a data monitoring framework that assists in complying with regulatory standards. Concord employs a "syndicated approach" where user's access to institution's data is dependent upon collective interaction of the entities that govern the data. The entities can only be partially trusted; while the complete trust can be placed only after their interaction.

Towards supporting such an interaction, Concord uses a 2-out-of-2 threshold cryptographic technique, which requires at least two entities to concur for data access. This paper uses the cryptographic technique called mediated RSA (mRSA) [6, 14]. Concord framework can be extended to use any 2-out-of-2 threshold cryptographic technique.

Using the Concord framework, a user of the mobile device needs to obtain the organization's consent to access data that resides on the mobile device. Consequently, upon detection of a compromise, the organization immediately discontinues further data access requests. An intruder would need to obtain both the organization's and the user's permission to access data. As Concord framework maintains the list of files accessed by users, it provides the organization with the knowledge of the exposed data.

Concord separates the data access history management from availability of data. As data availability can be improved by increasing the number of file servers, data access history should be maintained by another entity. Moreover the organization could securely maintain data access histories by securing a single entity.

The rest of the paper is organized as follows: the next section discusses the threat model addressed by Concord framework. Then related work in the area is overviewed after which the Concord framework and its design is shown. The next section describes the implementation issues of the Concord framework followed by an evaluation of the performance of the Concord framework and a conclusion.

Threat Model

This section discusses the threat model faced by institutions. Organizations usually maintain network file servers whilst, users carry a subset of the data on their laptops. Due to the scattering of data across different storage devices, the organization faces potential risks due to mismanaged data servers or mobile devices. Consequently, the organizations would like to maintain a history of accesses of critical data for the fear of improper use or compromise of the data which further helps towards complying with regulatory standards. Based on the risks faced, we classify the threats into three categories:

1. Insecure Data Servers
2. Loss of Mobile Device with/without Disk Encryption
3. Disgruntled Users

Insecure Data Servers

An institution's data server is susceptible to compromise due to mismanagement, improper configuration or worse, a hacker. If the data is not encrypted, a hacker can access and modify critical data. Data servers, hence, cannot be trusted to store unencrypted data as they can be prone to compromise. As institutions usually maintain highly available data servers with backups, we assume service disruption is not an issue, while storing encrypted data is of utmost importance.

Loss of Mobile Device and Disk Encryption Keys

Mobile devices, such as laptops, carry a part of the organization's data. Loss of the mobile device could imply a data loss. For example, mandatory regulation compliance standard such as the Feinstein Bill dictates that upon loss of critical customer data, organizations must communicate about the loss to the affected customers. In addition to loss, the data on the mobile devices is susceptible to risks arising from unauthorized access either due to spy-ware, malware or from unauthorized users.

An organization would benefit from the knowledge of subset of data loss enabling them to alert a subset of users rather than all. Upon identification of an unauthorized access, the organization should be precisely aware of the data that could have been exposed to potential risks. To secure against threats, organizations encrypt data on the hard disk and secure the data key. If a hardware-based data key is compromised, the data on the mobile devices is prone to unauthorized access.

Disgruntled Users

About 60% of the reported attacks have been either from current or former employees [8]. Yet, organizations enable personal laptops with critical data which could be compromised. Upon detection of such users, data access from the mobile device should be immediately revoked to disable further accesses to the data stored on the laptop. Further the access history should be known to the organization.

Related Work

Based on the threat model discussed in Section 2, we discuss other currently available systems. We categorize the related work into three sections:

- Data protection for Mobile Devices,
- Data Protection Frameworks, and
- the cryptographic techniques available for such a framework.

Data Protection for Mobile Devices

Toward securing a mobile device when the user is not in the vicinity, Transient Authentication [3, 4] introduces a mechanism whereby the data and the memory of the mobile device is encrypted. Only users who carry a hardware token can access the mobile device. Transient Authentication aims to defend a system against unauthorized physical access. However, Transient Authentication does not consider the security threats arising from within the organization.

Another technique to secure the data is to encrypt it using a hardware-based encryption. For example, Seagate's hardware-based key is a technique that encrypts the disk. Though an encrypt-on-disk mechanism protects data on the mobile device when it is stolen, it cannot protect against loss of the key or a disgruntled user. In case both the key and the disk are lost, the organization would not be aware of the compromised data on the disk. Therefore, hardware key based solution does not consider compliance with regulatory standards.

IBM tackles these security issues with their laptops using smart cards and biometric authentication [11]. IBM has developed a Trusted Platform Module that stores user passwords on a chip. Should a mobile device be stolen, Absolute Software Corporation's Computrace transfers data to a remote location and erases the hard disk. These mechanisms do not safeguard the data against malicious users.

Data Protection Frameworks

Plutus [9] provides strong security in an unsecured server (file server) setting. Plutus's design consideration introduces the concept of the data owner (reader and writer) who maintains the keys for the data. In this mechanism, the data creator is a trusted entity. Plutus uses a lazy revocation mechanism to revoke a users' data access, i.e., write access is revoked when there is an attempt to write. Plutus assumes that the data creators own the data. Plutus assumes the mobile device is secure and does not protect against theft of data.

Network file servers, such as Encrypted NFS [12] use OpenSSH for secure communication. Network file servers provide centralized storage architecture that allows a single point of revocation for future data accesses. Network file servers require users to connect to the server to access the data.

Threshold Cryptographic Algorithm

Threshold cryptographic algorithms have been used to provide efficient revocation [6, 10]. mRSA [6, 14] is a 2-out-of-2 threshold cryptographic algorithm that provides a single public key and two private keys. The mRSA threshold cryptographic algorithm allows a trust model with three entities: a) the client; b) the mediator; and c) the server. The server maintains the public key whereas the two private keys are distributed among the client and the mediator. This necessitates that both the client and the mediator to partake in a trusted relationship to decrypt the data.

We use the mRSA cryptographic mechanism in Concord's design to assist in its efforts to monitor the data access of the users. Additionally, the mRSA technique provides a mechanism for fast revocation of a mobile device.

We propose the Concord framework which places partial trust on all the entities (data server and mobile device). Concord provides a mechanism to monitor user activity by employing the mRSA cryptographic technique. The collective interaction of the entities ensures data protection without the need for any additional hardware. As the key distribution is managed by a single trusted infrastructure-level service, Concord enables the administrator to set security levels in accordance with the critical value of the data. We discuss the different security levels later.

Concord Framework

The Concord framework addresses the threat model discussed earlier by distributing the trust among multiple entities. Concord framework mandates that only encrypted data be available to any entity and access to data can only be permitted after a successful interaction among them. Concord employs both encrypt-on-disk and encrypt-on-wire mechanisms.

As Concord assumes that a laptop transmits the data over the Internet, we place a constraint on the laptop should be connected to the Internet to be able to access the data. We transfer the metadata rather than transmitting huge files. We assume that if decrypted data is locally-copied, data is lost.

As organizations might have multiple file services, Concord minimizes the data access control management by segregating the data management and access control to a single entity rather than on multiple file servers.

Concord assumes that the data creator need not be the data owner. The data access policies, therefore, need to be provided by the organization. The infrastructure-level monitors and revokes data accesses on a user's mobile device. As discussed earlier, along with the need to encrypt data for storage and transfer, the interaction with an intermediate entity mandates the use of a cryptographic mechanism. The cryptographic algorithm should:

1. Immediately revoke the client.
2. Mandate collaboration among entities for data access. (This ensures that compromise of any single entity does not compromise the data – by distributing trust among multiple entities.)

A 2-out-of-2 cryptographic mechanism supports such a requirement. The cryptographic mechanism requires two entities to collaborate to decrypt the contents. This provides an infrastructure-level service capable of maintaining access patterns of the users. We use the mediated RSA cryptographic algorithm [6, 14] a 2-out-of-2 cryptographic technique. In the following section, we discuss the mRSA protocol.

mRSA Protocol

The Mediated RSA (mRSA) cryptographic protocol consists of a public key associated with two private keys. The private keys are distributed to one of the two entities: the Security Mediator (SEM) and the client. Any content that is encrypted by the server (holder of the public key) can be decrypted by combining decryptions by both the mediator and the client. Any single entity is unable to decrypt the content independently.

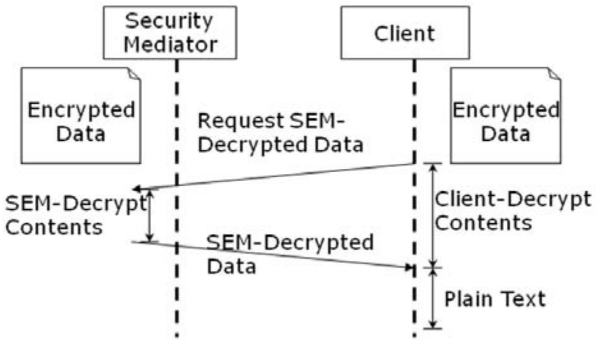


Figure 1: mRSA protocol with interaction between the mediator and the client to decrypt data (SEM – Security Mediator).

Figure 1 demonstrates the mRSA protocol. The encrypted contents are stored both at the client and the SEM. When the client requires access to the unencrypted contents, the client requests the SEM to decrypt the contents with its private key and starts decrypting the content with its own private key and is referred to as SEM-decrypted or Client-decrypted data. A single decrypt operation by Security Mediator does not provide plain text. The SEM-decrypted content is still encrypted with RSA strength encryption. The mediator transmits the SEM-decrypted data to the client. The client combines the SEM-decrypted data with the Client-decrypted data to access the plain text. If the client is revoked, the mediator does not compute or transmit SEM-decrypted contents. Without the mediator's participation, the client cannot compute the plain text.

Employing the mRSA algorithm directly to mobile networks to allow them to share a huge size of data creates performance issues. For instance, if the encrypted data is about 1 GB, employing mRSA requires sizable bandwidth to transfer the data between the mediator and the client. To overcome this limitation, Concord framework uses the mRSA protocol to transfer secret key algorithm for cryptographically securing the data.

Concord Components

In this section, we describe the various components of the Concord Framework. Concord assumes that the infrastructure secures and maintains infrastructure-based entities.

Figure 2 shows the different components of Concord. The trusted entities are:

1. Trusted Key Server
2. Connected Enforcer

And un-trusted entities include:

3. Disconnected Enforcer
4. Data Server and
5. Mobile Device

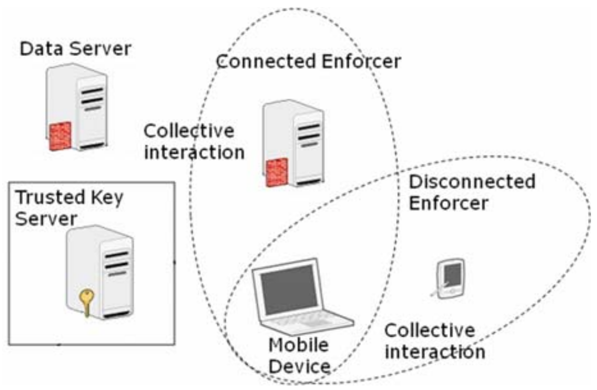


Figure 2: Concord Components – Data Server, Disconnected Enforcer (D-Enforcer) and Mobile Device are un-trusted where as the Key Server and the Connected Enforcer (C-Enforcer) are completely trusted. The collective interaction among one of the enforcers and the Mobile Device can decrypt the data.

Trusted Key Server

The Trusted Key Server generates cryptographic keys for other components and for the data. To provide higher security, Concord partitions the data into blocks referred to as Data Units. Each Data Unit is encrypted with a secret key algorithm using Data Unit Keys. A Data Unit can be the entire disk volume, a directory or individual files. A flexible Data Unit allows multiple security levels depending upon data sensitivity. The advantage of the design allows a compromise be confined to a single unit. The Trusted Key Server stores the Data Unit Keys for other entities.

In addition to Data Unit Keys, the Trusted Key Server creates mRSA keys. When a new mobile device joins the organization, the mobile device creates an mRSA keyset. The Trusted Key Server stores the public key and passes the other keys to the enforcer and the client. The Trusted Key Server creates a secondary multiple mRSA key pair if a Disconnected Enforcer is involved.

Connected Enforcer

The Connected Enforcer (C-Enforcer) is a trusted infrastructure entity where the Data Unit access policies are enforced. The C-Enforcer is available over either wired or wireless networks. As shown in Figure 2, the mobile device can access the plain-text only after interacting with a C-Enforcer. If the mobile device is revoked, C-Enforcer does not provide the SEM-decrypted Data Unit Keys, disabling the mobile device's ability to view plain text.

C-enforcer has an additional benefit: it can maintain the list of Data Units accessed by a mobile device. Due to this benefit, the organization determines a user's data access patterns as and when they request Data Unit Keys. This determines the subset of data that has been accessed by the users of mobile devices. In the event of a security breach, user access history is critical for an organization to comply with regulatory standards. We assume that the C-Enforcer stores the logs in a tamper-proof storage.

C-Enforcer and Trusted Key Server can both be a part of the same entity or can be separated. We design the two as different entities to demonstrate the different functionality.

Disconnected Enforcer

To support mobility, Concord supports a Disconnected Enforcer (D-Enforcer) that functions as an enforcer in the absence of the C-Enforcer. The D-Enforcer, however, is an un-trusted entity that caches only a part of the Data Unit Keys to enables local reads and writes. The data on the D-enforcer is not monitored as rigorously as C-enforcer as: a) it does not maintain data request logs; and b) any key revealed to it is considered to have been viewed by the users.

Storing the SEM-decrypted content along with encrypted data may lead to data and key compromise. Thus, D-enforcer (e.g., a PDA) provides a second layer of security when data is stored on a mobile system. We assume that a mobile device is a laptop and the D-Enforcer usually is a PDA. By mandating the use of a D-Enforcer, the risk of data loss is distributed among multiple entities as the loss of both is less likely.

Data Server

The Data Server stores the encrypted data and can function with any file system. Concord assumes that the Data server is not a trusted entity and that it can be replicated to provide high availability. By separating data access from data governance, organizations can provide highly replicated service while minimizing the overhead of access control. Storing encrypted Data Units on an infrastructure's data servers provides a two-fold advantage:

- the Data Server compromise does not imply data compromise; and
- the data is available to the users upon the loss of the Mobile Device.

Mobile Device

A mobile device, such as a laptop, maintains the data in an encrypted format to secure data from physical loss. The encrypted Data Unit Keys are stored on the Mobile device as well as on an Enforcer. The Data Unit Keys can be transmitted securely using the mRSA protocol. The Data Unit Keys should be securely handled on the Mobile Device and deleted when the need for unencrypted data no longer exists.

Concord Protocol

Concord provides a mechanism to download and store encrypted data on the Mobile Device. In a connected mode, the Mobile Device interacts with the C-Enforcer; in the absence of the C-Enforcer, the D-Enforcer doubles up as an enforcer. Both the Enforcers have different mRSA key pairs as the C-enforcer' keys are organizational as compared to the D-Enforcer.

As explained in the above sections, each Data Unit is encrypted with a Data Unit Key generated using a symmetric key algorithm. The use of the symmetric key algorithm avoids transferring huge data using the mRSA algorithm. This subsection discusses the design of Concord, that is, mRSA key setup, reader and writer architecture using the C-Enforcer and the D-Enforcer.

Key Setup

In the Concord framework, key setup is required in two cases: first, when a new Mobile Device joins the Concord framework; second, when new Data Unit Keys are transferred to the Mobile Device. As mentioned in previous sections, the Concord framework uses mRSA, 2-out-of-2 threshold cryptography.

mRSA Key Setup

Figure 3 shows the process of setting up the mRSA keys for a Mobile Device. When a new Mobile Device joins the Concord Framework, the Mobile Device requests the Trusted Key Server for mRSA keys to be generated. The Trusted Key Server performs a validation check to ensure that the client has not been previously revoked by checking a revocation list. If the Mobile Device has been revoked, the mRSA keys are not generated. On the other hand, if the Mobile Device has not been revoked, mRSA keys are generated for the client. If the C-Enforcer is involved, the mRSA keys are transmitted to the Mobile Device and the C-Enforcer. If the D-Enforcer is required in addition to the C-Enforcer for a Mobile Device, the mRSA keys are communicated securely to the C-Enforcer, the D-Enforcer and the Mobile Device. The creation of an mRSA key can occur only after an mRSA key is obtained for the C-Enforcer.

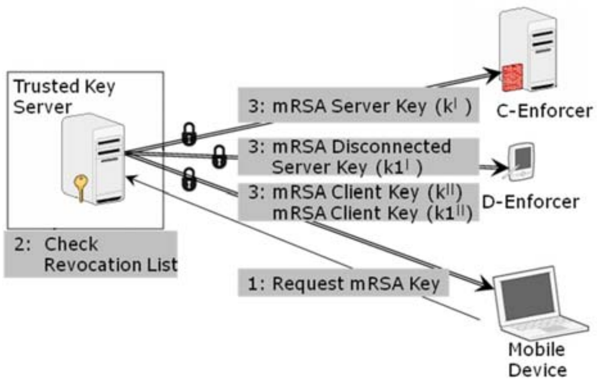

Trusted Key Server
3: mRSA Server Key (k^{I})
C-Enforcer
3: mRSA Disconnected Server Key ($k1^{I}$)
3: mRSA Client Key (k^{II}) mRSA Client Key ($k1^{II}$)
D-Enforcer
2: Check Revocation List
1: Request mRSA Key
Mobile Device

Figure 3: mRSA Key Setup for a new Mobile Device. When a Mobile Device requests for mRSA Key, the Trusted Key Server checks the revocation list. If the client is revoked, it does not receive the mRSA keys else the mRSA keys are transmitted to other entities.

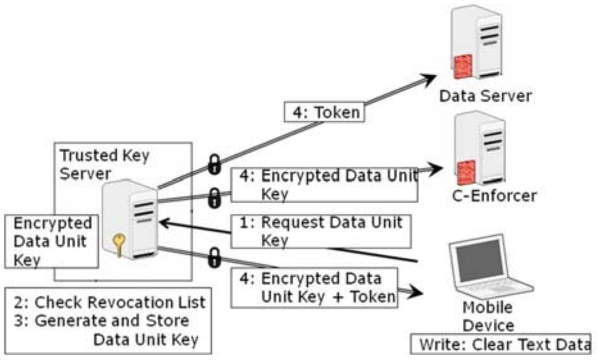


Figure 4: Obtaining Data Unit Keys. The Mobile Device requests the encrypted Data Unit Keys for the reader architecture. In the writer architecture, the Trusted Key Server provides the Mobile Device with a new Data Unit Key.

Data Unit Keys

Figure 4 demonstrates the mechanism for a Mobile Device to obtain a new Data Unit Key. When a Mobile Device requests the Trusted Key Server for a new Data Unit Key, the Trusted Key Server performs validation checks and generates the keys. The Data Unit Key is stored on the Trusted Key Server and the encrypted Data Unit Key is transmitted to the C-Enforcer and Mobile Device. In addition, the Trusted Key Server creates a token which is transmitted to the Mobile Device and the Data Server. When the Mobile device wishes to put the encrypted Data Unit on the Data Server, the Data Server verifies the token and stores it.

Download Encrypted Data Units

Concord can be configured for use with any underlying file system. As discussed in the above sections, the Data Server stores encrypted Data Units. We assume that the underlying file system provides basic access control allowing only authorized users to download the encrypted data. Once the user is authenticated, the encrypted data can be downloaded to a Mobile Device and stored for future access.

Read and Write Enforcement Protocol Using the C-Enforcer

When a user creates data, Concord provides a secure mechanism to upload the Data Unit to the Data Server. Assuming that the mRSA keys have been assigned, upon the request for a Data Unit Key, the Trusted Key Server communicates the encrypted Data Unit key to the C-Enforcer and the Mobile Device.

Figure 5 and Figure 6 shows the process of decrypting the Data Unit key through the interaction between the Mobile Device and the C-Enforcer. When the Mobile Device requests the decryption of the Data Unit key, the C-Enforcer checks the revocation list. Upon successful validation, the C-Enforcer decrypts the Data Unit Key using the mRSA private key it holds. The SEM-decrypted Data Unit Key is then transmitted to Mobile Device. The Mobile Device simultaneously decrypts the encrypted Data Unit key using its own private key. Finally, the Mobile Device combines the SEM-decrypted and Client-decrypted Data Unit Keys to retrieve the Data Unit Key. In the reader architecture (Figure 5), the client is able to read the content in the encrypted Data Unit using the Data Unit key obtained.

In the writer architecture shown in Figure 6, the Mobile Device requests for a Data Unit Key to encrypt the data. Once the data is encrypted, the user places the data onto the data server. If the Mobile Device requests to store the data as a part of a new Data Unit, it requests from the Trusted Key Server a single Data Unit Key that can be used to encrypt the clear data. A new Data Unit Key is created based on the protocol described earlier. When storing the Data Unit, the Data Server checks the randomly generated token. Token verification is used to indicate that a new Data Unit has been created and disallows creation of Data Units with the same name and location. Concord allows data writes only when the Mobile Device is in a connected mode. We explain the disconnected mode in the next section. In comparison, if the data needs to be added to an existing Data Unit, the Mobile Device retrieves the Data Unit Key, encrypts the Data Unit with the Data Unit Key, and stores it on the Data Server.

Reader Architecture for D-Enforcer

In a disconnected mode, the C-Enforcer delegates data enforcement to the D-Enforcer. In comparison to the C-Enforcer, the D-Enforcer maintains a subset of Data Unit Keys. We believe that the subset of keys available to the D-Enforcer can be determined by an organizational policy regarding the number of Data Units to be shared based on the data sensitivity. Such a policy does not necessarily safe-guard against a hacker for which the keys are available on the D-Enforcer, it only identifies this data. We discuss key policy in the next section.

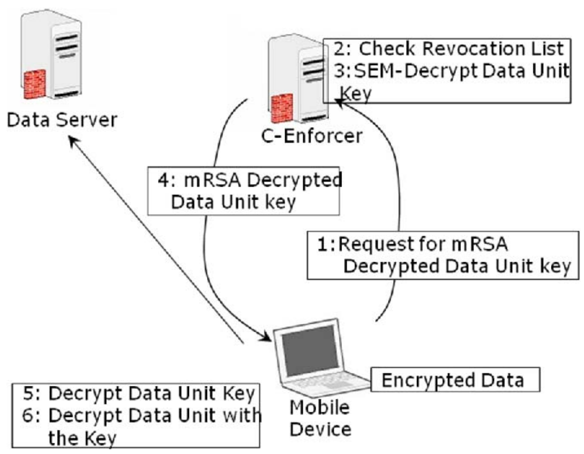

Data Server
C-Enforcer
2: Check Revocation List
3:SEM-Decrypt Data Unit Key
4: mRSA Decrypted Data Unit key
1:Request for mRSA Decrypted Data Unit key
Encrypted Data
5: Decrypt Data Unit Key
6: Decrypt Data Unit with the Key
Mobile Device

Figure 5: Decrypting the Data Key to accessing a Data Unit Key for reader architecture.

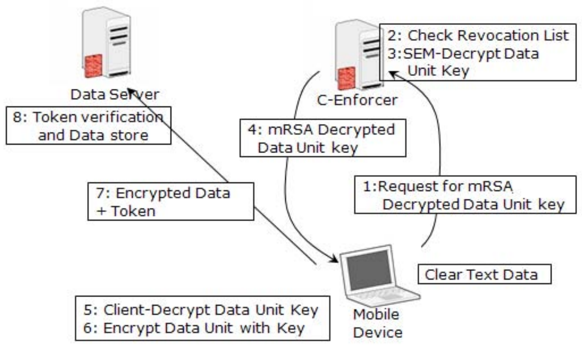

Data Server
8: Token verification and Data store
C-Enforcer
2: Check Revocation List
3:SEM-Decrypt Data Unit Key
4: mRSA Decrypted Data Unit key
7: Encrypted Data + Token
1:Request for mRSA Decrypted Data Unit key
Clear Text Data
5: Client-Decrypt Data Unit Key
6: Encrypt Data Unit with Key
Mobile Device

Figure 6: Decrypting the Data Key to encrypt data for writer architecture.

Figure 7 shows the Mobile Device requesting a subset of Data Unit Keys from the Trusted Key Server. The Trusted Key Server performs the validation checks and transfers the complete set of Data Unit Keys to the Mobile Device and the C-Enforcer. In contrast to the C-Enforcer, the D-Enforcer and the Mobile Device receive a subset of the Data Unit Keys. The Mobile Device maintains two sets of encrypted Data Unit Keys: one encrypted with the mRSA key of the C-Enforcer and the other with that of the D-Enforcer.

Upon request from the Mobile Device for a Data Unit key, the D-Enforcer decrypts the key and transmits the SEM-decrypted key to the Mobile Device. The Mobile Device uses the SEM-decrypted key to compute the complete Data Unit Key. For regulatory compliance, all the data whose Data Units keys were given to D-Enforcer are assumed as accessed.

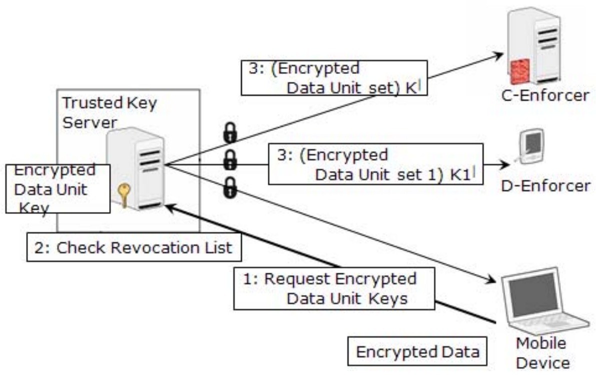


Figure 7: The Trusted Key Server distributes the Data Unit Keys to the Enforcers. The C-Enforcer maintains the complete set of Data Unit Keys whereas the D-Enforcer maintains smaller subset of Data Unit Keys.

Security in Concord

In this section, we discuss the security provided by Concord. We primarily discuss: (a) revocation of a Mobile Device; (b) granularity of the Data Unit and (c) D-Enforcer Data Unit Key subset policy. Table 1 discusses how Concord secures data in case of different compromises. We assume that the data on the mobile device is decrypted in a secure location. Concord cannot secure data that has been decrypted and copied onto devices that it cannot monitor. Upon loss of the Mobile Device a part of the keys along with the data would be available to the attackers. However, the data is encrypted and the keys are still encrypted. To decrypt the key, the second private key from an Enforcement Point is required. If the attacker has the D-Enforcer, a part of the keys could be decrypted and the other part of data is provably secure. The other option would be attack the C-Enforcer, which would be monitored. Therefore, without an Enforcer, the data is secure.

Revocation in Concord

Concord supports: a) revocation of read access to previously unread Data Units and b) revocation of the write access to all Data Units. If a Mobile Device needs to be revoked following a compromise, administrators can perform the revocation at the Trusted Key Server. The Trusted Key Server securely transfers the revocation list to the C-Enforcer immediately upon revocation. Further data access to the unread data on the affected Mobile Device is not possible as the C-Enforcer will not provide the Data Unit Keys to the Mobile Device.

In the disconnected mode, the D-Enforcer stores a subset of the Data Unit Keys. A key distribution policy, enforced at the Trusted Key Server, determines the number of keys that can be stored at the D-Enforcer for Mobile Devices. The D-Enforcer stores a smaller subset of keys that can be used to decrypt the Data Units even after revocation of the Mobile Device. However, as the users cannot access the other Data Unit Keys, the compromise is contained. Future versions of the Data Units are encrypted with new Data Unit Keys, to protect them from future read access.

When D-Enforcer needs new keys, the old set of keys that was cached should be cleared and a new set can be provided. In such a case, the old set of Data Units is assumed to have been exposed to the user.

Granularity of the Data Unit

Partitioning the Data into multiple Data Units determines the desired security level when employing the Concord framework. When a potential compromise occurs or the data needs to re-encrypted following a user revocation, data partitioning allows re-encryption of a smaller subset of the data, instead of encrypting, say, the entire volume. The granularity of the Data Units can be configured by the organization and will dictate the level of security. The three available granularities for Data Units are:

- File-granular;
- Directory-granular; and
- Volume-granular.

File-granular configuration creates a Data Unit key for each file. This is the highest level of security that requires a large number of keys. For example, our experiment shows that for 77,900 files, about 3 MB is required to store the keys. We believe that this is acceptable as both the C-Enforcer and the Trusted Key Server are dedicated systems used to store keys for multiple Mobile Devices.

Directory-granular configuration provides security by encrypting the files in a particular folder; however, the sub-folders are encrypted using another Data Unit key. This granularity provides lesser security compared to the File-granular configuration, as multiple files share the same key. For the above example, we note that there were about 7,251 folders taking up about 0.3 MB.

Threat	How Concord helps
1. Insecure Data Server	Data on the Data Server is secure - as it is encrypted
2. Loss of Mobile Device	Data on Mobile Device is secure - as it is encrypted
3. Loss of D-Enforcer	Data Keys is secure - as it is encrypted
4. Loss of both Mobile Device and D-Enforcer	List of Compromised Data is available. Rest of the data is provably secure.
5. Disgruntled Users	List of Data accessed is available

Table 1: How Concord is able to secure the data for different compromises.

The volume granularity configuration involves use of a single key for the entire volume, providing the least security available with Concord. Such a system is similar to encrypt-the-disk system.

Each level of security comes with the cost of encryption and decryption. For example, if an organization wants to maintain a high level of security, it would imply that the organization needs to employ File-granularity, which would result in high costs in terms of storage and encryption and decryption operations. On the contrary, if the organization would like to maintain minimal security, the volume-granular can be used, which requires a minimal amount of time to decrypt the data. We suspect that a reasonable balance between security and cost would be to employ directory-granularity for the data-units.

D-Enforcer Data Unit Key Subset Policy

As discussed in previous sections, the number of encrypted Data Unit Keys cached by the D-Enforcer is determined by organizational policy. Upon successful transmission of these keys to the D-Enforcer, the Trusted Key Server transmits the list of keys revealed to the C-Enforcer for regulatory compliance. For all the Data Units for which the encrypted Data Unit Keys are revealed to D-Enforcer, the user is assumed to have accessed the data for regulatory compliance. Therefore, a policy to restrict the amount of files that users can access in a disconnected mode is required to provide highly secure systems. For example, if a disgruntled user is known to have checked out a huge number of Data Unit Keys over time, it would imply that a huge amount of data might have been compromised. We suggest that such a policy should be designed on a need-to-know basis.

Concord Implementation

In this section we discuss the implementation details of each of the components of the Concord framework.

Trusted Key Server Implementation

The Trusted Key Server requires administration to set the levels of security. For instance, apart from revocation, Trusted Key Server allows Data Unit key generation and distribution. All key-requests mandate Trusted Key Server to push keys to C-Enforcer or D-Enforcer. The revocation list at the Trusted Key Server should be kept updated and updated at C-Enforcer to discontinue further key requests.

The Trusted Key Server serves the following requests:

- The mRSA key setup, including checking the revocation list and generation of unique mRSA key pairs for both the Enforcers and the Mobile Device.
- Generation of the Data Unit Keys using the Advanced Encryption Standard (AES) standard.

The Trusted Key Server stores the mRSA keys for each Mobile Device. The Trusted Key Server maintains the list of revoked Mobile Devices. Further, the Trusted Key Server maintains the list of the Data Units and their corresponding Data Unit Key for each Mobile Device.

C-Enforcer Implementation

The C-Enforcer's communication with the Mobile Device is the primary mode of request for a Data Unit key that requires mRSA decryption. The C-Enforcer has been implemented in Java. The communication of the C-Enforcer with various components has been developed using the Java's socket library. The current Concord prototype allows only a single Data Unit Key to be decrypted per request.

The encrypted Data Unit Keys are stored on the Trusted Key Server as a single data structure (a Java HashMap). The data structure keeps a mapping between the hash of the Data Unit's storage path and the associated Data Unit Keys. We implemented this feature in Concord using SHA1 that generates a 20 byte output.

D-Enforcer Implementation

We implemented the D-Enforcer using Java ME running on a PDA. The Java ME Virtual Machine (JVM) was CrEme 4.0 [5] compliant with J2ME Connected Device Configuration (CDC) 1.0 specification based on JDK 1.3.1. The D-Enforcer stores encrypted Data Unit Keys in a Java HashMap. Storing hash of the data as an identifier to associate the encrypted file keys reduces the in-memory data structure size on D-Enforcer.

For the implementation, D-Enforcer has two services on the active port in a) receiving and b) sending mode. The communication between the Trusted Key Server and the D-Enforcer is in the receiving mode; allowing the data to be pushed from the Key Server to the D-Enforcer as and when required. The communication between the D-Enforcer and the Mobile Device is in the sending mode; allowing the D-Enforcer to push the data to the Mobile Device. Such a mechanism has been designed to reduce the amount of services run on the D-Enforcer to reduce the cost of operations on the D-Enforcer.

Data Server Implementation

The Data Server prototype has been implemented using Java. The functionality can be easily provided using any file system such as NFS or AFS. The Data Server interacts with the Mobile Device over two channels: the Control Channel and the Data Channel. The Control Channel serves the specific goal of controlling the data that needs to be transferred where as the Data Channel is used to transfer itself. The Control Channel sets up the server socket for the Data Channels to open multiple sockets for streaming. The socket implementation supports a large number of files that are streamed in parallel. The streaming mechanism employs a thread pool to stream one file per thread.

Mobile Device Implementation

The Mobile Device is a temporary store for encrypted Data Units. We implement the Mobile Device interface using Java. We designed a command line interface which allows the users to register, login, conduct mRSA key setup, request encrypted data, decrypt content and lastly, logout.

Experiments and Results

This section discusses Concord's performance. The performance analysis involves (a) Performance comparison of the C-Enforcer versus that of the D-Enforcer; (b) Data Unit Key Generation time; (c) Data Unit Key Distribution time for both the C-Enforcer and the D-Enforcer.

Experiment Setup

All of the experiments have been performed using the following devices and networks with specified configurations. The Trusted Key Server was running on Intel Pentium 4 CPU 3.00 GHz with 2 GB RAM running Microsoft Windows XP Professional version 2002 with service pack 2, whereas the C-Enforcer and Data Server were running on different machines with a similar configuration. The Mobile Device used for performance analysis was a Intel Pentium M 1700 MHz 1 GB RAM whereas the D-Enforcer was a Intel ARM HP iPAQ PocketPC h4300 PDA with 64 MB RAM running Windows CE.net Version 4.1. The Local Area Network bandwidth was 100 Mbps with a delay of about 0.1-0.2 milliseconds and Bluetooth link was 700 Kbps with a delay of about 60-70 milliseconds. The Java Virtual Machine (JVM) on the PDA is CrEme 4.0 [5] compliant with J2ME Connected Device Configuration (CDC) 1.0 specification and based on JDK 1.3.1. The CDC is the J2ME configuration that supports full Java implementation on PDA HP iPAQ PocketPC.

Performance Comparison Between C-Enforcer and D-Enforcer

As indicated above, the D-Enforcer communicates with the Mobile Device over a low-bandwidth Bluetooth communication link. On the other hand, a C-Enforcer is implemented over a high bandwidth connection and has additional computational resources.

Figure 8 shows the total decryption time for a single Data Unit Key. The total time includes the time taken to partially decrypt the Data Unit Key on the C-Enforcer and on the D-Enforcer; finally, the partially-decrypted data from the Enforcer point and the Mobile Device is used to completely decrypt the Data Unit Key. The performance of D-Enforcer is improved by reusing the socket that is used through the socket pooling. The D-Enforcer shows significant performance improvement when the socket is reused. The average one-time socket connection setup time was found to be 4.39 seconds. Since the D-Enforcer is dedicated to serving a single Mobile Device, the communication socket can always remain open. That is, no socket connection timeout is set.

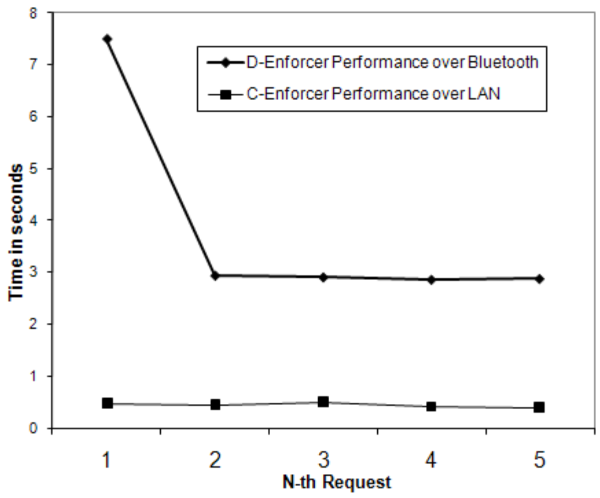


Figure 8: Performance analysis of D-Enforcer and C-Enforcer involves the measurement of the time taken to decrypt a Data Unit Key (48 bytes) against the nth decryption request. The average one-time socket connection setup time for Bluetooth was found to be 4.39 Seconds.

Concord Setup Performance

Concord setup requires time to distribute Data Unit Keys (AES keys) to the Mobile Device and the C-Enforcer as well as the D-Enforcer. This section shows (a) Trusted Key Server performance and (b) the performance of file key distribution to the Enforcers.

Trusted Key Server Performance

The time taken to generate Data Unit Keys is proportional to the number of keys that are being generated; however, as the number of keys increases, the time taken to generate the keys is relatively constant. Our experiments show that the time required for generation of a key decreases exponentially as the number of keys generated increases. When one key was generated, 0.46 Sec per key was required whereas 1.06 Sec per key was required when 1,000 keys were generated. We provide an optimization for Trusted Key Server performance by creating multiple keys. These keys are provided to the users upon request.

Performance of Data Unit Key Distribution to Enforcers

In Concord, Data Unit Keys are transmitted from the Trusted Key Server to the Enforcers and to the Mobile Device. The amount of time taken to send the Data Unit Keys from the Trusted Key Server to the Enforcers is shown in Figure 9. For the D-Enforcer, using Bluetooth, our experiments show that about 4.39 seconds are required to create a socket. The time taken to transmit about 1000 keys is about 6.5 seconds.

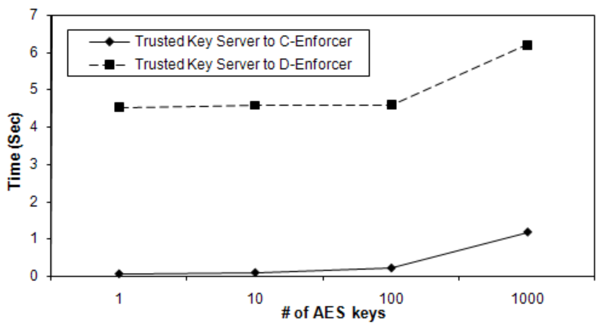


Figure 9: AES Key distribution time from Trusted Key Server to D-Enforcer and C-Enforcer.

The experiment reiterates the fact that the time taken to communicate the Data Unit Key is dependent on the bandwidth of the communication medium. The Bluetooth link is a very low bandwidth in comparison to that of LAN.

Conclusion

Concord framework allows organizations to monitor the data accessed on a mobile device as an infrastructure-level service by distributing the trust among multiple entities. With an ability to provide irrefutable data access history, Concord data access framework supports regulatory compliance standards. The knowledge of the data accessed by the users on their Mobile Device enables the organization to demonstrate their adherence to mandated regulatory compliance such as HIPPA and the Feinstein Bill.

Concord provides a novel secure data access and monitoring framework with data being stored on the employee's mobile devices. Concord requires an enforcement server, either the connected C-Enforcer or the disconnected D-Enforcer, to be involved in decrypting the data on user's mobile device. With the help of the enforcement server, Concord enables the organization to effectively monitor data access and revoke unwanted clients. Concord identifies critical organizational data that has been accessed by a disgruntled user's access and theft or loss of the mobile device by handling data securely during storage and wire while in a cryptographic format. Concord employs a 2-out-ot-2 cryptographic technique, called mRSA, which encrypts data while in storage or when being transferred on wire.

Future Work

The Concord implementation discussed in this paper has been implemented as a prototype. During this prototype implementation, our measurements demonstrated that the high cost involved in encryption and decryption using mRSA. We are working on another threshold cryptographic solution that would be able to reduce the number of operations to increase the efficiency. We plan to perform an extensive I/O evaluation to evaluate the costs involved.

Acknowledgments

This research is partly supported by TIAA CREF Biggs Faculty Fellowship. We also would like to thank Pratik Thanki for his help in setting up the experiment.

Author Biographies

Gautam Singaraju is a doctoral candidate at University of North Carolina at Charlotte. As part of the research efforts, he focuses on secure IT infrastructure design; specifically on system administration issues. Some of his work is in: 1) securing email infrastructure, 2) infrastructure design for regulatory compliance – premise-aware data protection services, 3) intrusion detection systems, and 4) Virtualization-based secure infrastructure. Gautam can be reached at gsingara@uncc.edu .

Brent Hoon Kang is currently an assistant professor at the College of Computing and Informatics at UNC Charlotte. He leads the Infrastructure Systems Research Lab at UNCC which explores the secure architecting of large-scale infrastructure systems. Through the lab, he has worked on (1) securing email infrastructure, (2) research on malware and botnet enumeration and (3) topics such as premise-aware data protection infrastructure, and IT infrastructure design for regulation compliance. Recently he has been working with a group of IA students in researching malware and bot infection behavior as part of the Global Honeynet Research Alliance. As part of his efforts on Information Assurance (IA) education program, he has been developing the hands-on cyber exercise components that foster students' creativeness and problem solving skills for IT systems design and defense. He received his Ph.D. from the University of California at Berkeley. Hoon can be reached at bbkang@uncc.edu .

Bibliography

[1] Bluetooth Wireless, "Bluetooth Wireless Technology," *Bluetooth SIG – The official website for the Bluetooth short range wireless connectivity standard*.

[2] Bluetooth Protocol and Bluetooth PAN (Personal Area Network), *Bluetooth Special Interest Group (SIG)*.

[3] Corner, M. D. and B. D. Noble, "Zero-Interaction Authentication," *Proceedings of MOBICOM*, Atlanta, GA, September, 2002.

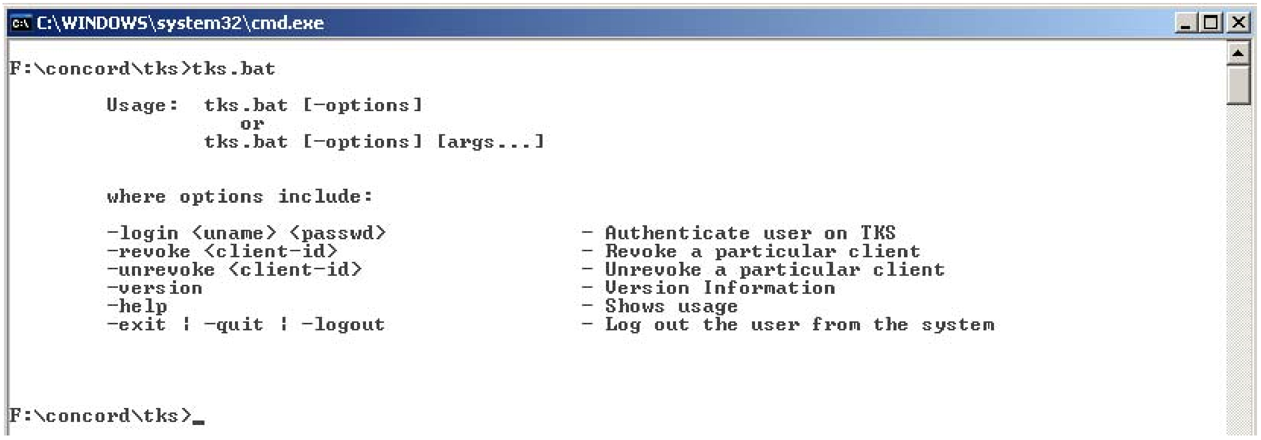
[4] Corner, M. D. and B. D. Noble, "Protecting Applications with Transient Authentication," *Proceedings First International Conference on Mobile Systems, Applications and Services*, 2003.

[5] CrE-ME 4.0 (J2ME-CDC) Beta, Java Virtual Machines for Java Based Embedded Devices, *J2ME CDC 1.0 Specification*.

[6] Ding, X. and G. Tsudik, "Simple Identity-Based Cryptography with Mediated RSA," *Proceedings of CT-RSA '03*, LNCS 2612, pp. 193-210, Springer, 2003.

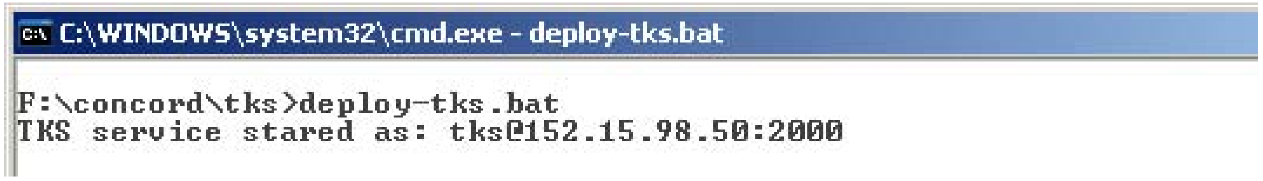
[7] Feinstein, Bill, "E-Loan and ING Direct Endorse Feinstein Identity Theft Legislation, First Two Major Companies to Endorse Feinstein Bill," June, 2005.
[8] Kabay, M. E., "Insider Attacks Are a Thorny Problem, Insider Computer Crime is Difficult to Defend Against," *Network World*, August, 2003.
[9] Kallahalla, M., E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus – Scalable Secure File Sharing on Untrusted Storage," *Proceedings Second USENIX Conference on File and Storage Technologies (FAST)*, USENIX, March, 2003.
[10] Libert, B. and J. Quisquater, "Efficient Revocation and Threshold Pairing Based Cryptosystems," *Proceedings 22nd Annual Symposium on Principles of Distributed Computing PODC '03*, July, 2003.
[11] Mitchell, R. L., "Decline of the Desktop," Computer World New Story, *Computer World*, September 26, 2005.
[12] Strandboge, J., "Encrypted NFS with OpenSSH and Linux," *SysAdmin Journal for UNIX and Linux System Administrators*, March, 2002.
[13] Standards for Privacy of Individually Identifiable Health Information, *Security Standards for the Protection of Electronic Protected Health Information*, April, 2003.
[14] Vanrenen, G. and S. W. Smith, "Distributing Security-Mediated PKI – Public Key Infrastructure," *EuroPKI 2004*, Springer-Verlag, LNCS 3093, pp. 218-231, June, 2004.
[15] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 21, Num. 2, pp. 120-126, February, 1978.

Appendix



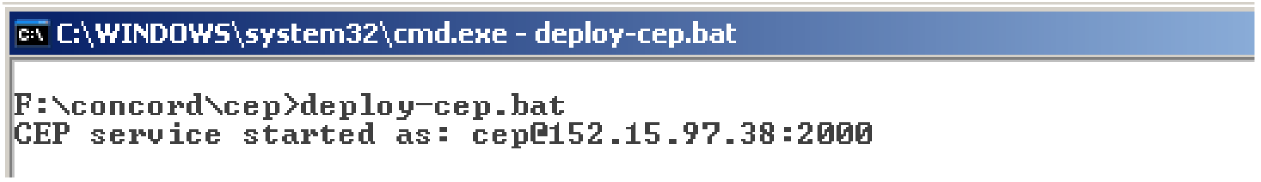
```
C:\WINDOWS\system32\cmd.exe

F:\concord\tks>tks.bat

        Usage:  tks.bat [-options]
                   or
                tks.bat [-options] [args...]


        where options include:

        -login <uname> <passwd>                 - Authenticate user on TKS
        -revoke <client-id>                     - Revoke a particular client
        -unrevoke <client-id>                   - Unrevoke a particular client
        -version                                - Version Information
        -help                                   - Shows usage
        -exit | -quit | -logout                 - Log out the user from the system



F:\concord\tks>_
```

Snapshot I: The TKS interface for administrators allows the revocation of malicious client.



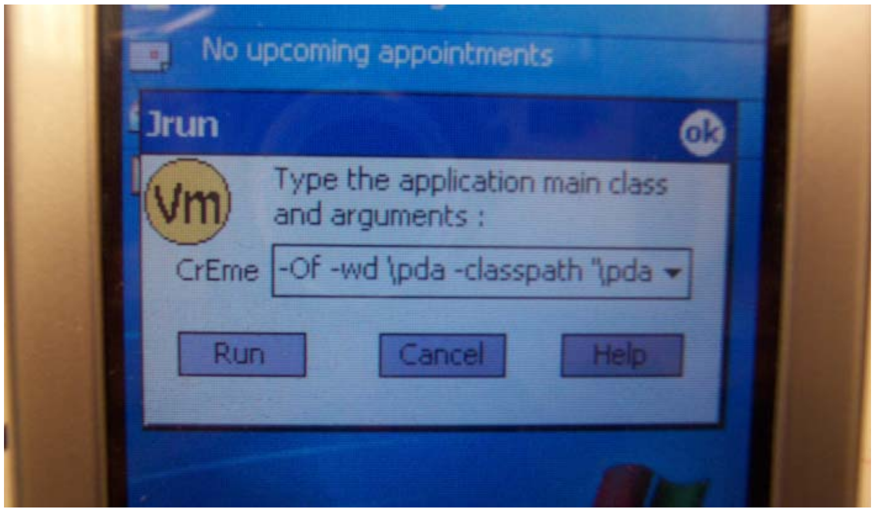
```
C:\WINDOWS\system32\cmd.exe - deploy-tks.bat

F:\concord\tks>deploy-tks.bat
TKS service stared as: tks@152.15.98.50:2000
```

Snapshot II: The Command that starts the TKS server.



```
C:\WINDOWS\system32\cmd.exe - deploy-cep.bat

F:\concord\cep>deploy-cep.bat
CEP service started as: cep@152.15.97.38:2000
```

Snapshot III: Command that starts the CEP server.

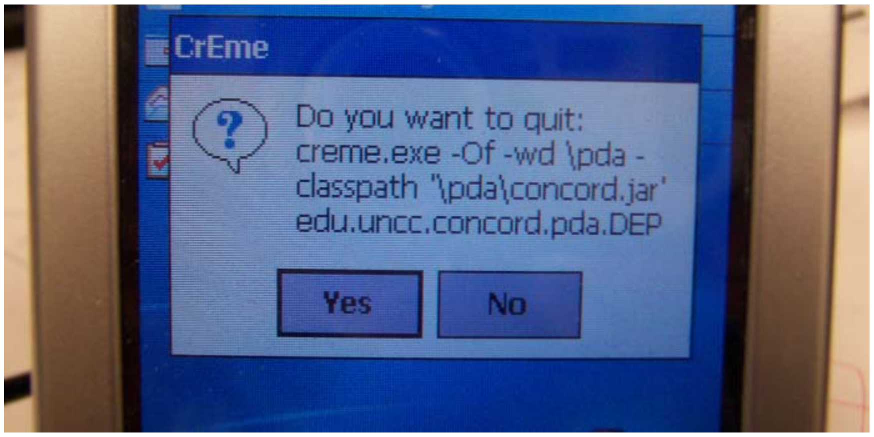

No upcoming appointments

Jrun

Type the application main class and arguments :

CrEme -Of -wd \pda -classpath '\pda

Snapshot IV: Command that starts the DEP server on a mobile device using CrEme jvm. The command is: CrEme.exe -Of -wd \pda -classpath '\pda\concord.jar' edu.uncc.corcord.pda.DEP.


CrEme

Do you want to quit: creme.exe -Of -wd \pda -classpath '\pda\concord.jar' edu.uncc.concord.pda.DEP

Snapshot V: The dialog box message that gets displayed when the DEP server is running on a mobile device.

Authentication on Untrusted Remote Hosts with Public-key Sudo

Matthew Burnside, Mack Lu, and Angelos D. Keromytis – Columbia University

ABSTRACT

Two common tools in Linux- and UNIX-based environments are SSH for secure communications and sudo for performing administrative tasks. These are independent programs with substantially different purposes, but they are often used in conjunction. In this paper, we describe a weakness in their interaction and present our solution, public-key sudo.

Public-key sudo[1] is an extension to the sudo authentication mechanism which allows for public key authentication using the SSH public key framework. We describe our implementation of a BSD SSH authentication module and the SSH modifications required to use this module.

Introduction

In today's age of large, distributed networks, trusted remote machines are rare. That is, untrusted machines are the common case, but through business or other requirements, users and administrators find themselves required to connect to such machines, regardless. These may be machines maintained by disreputable system administrators, machines which are believed to have suffered compromises, or simply machines for which the user suspects there is a high probability of future compromise. It is desirable not to provide sensitive information to such machines.

Two tools which have become the norm in such environments are SSH [15] and sudo [11]. These are indpendent programs with substantially different purposes, but they are often used in conjunction. In this paper, we describe a weakness in their interaction, and then present public-key sudo to solve it.

SSH is a tool used for secure communication between computer systems. It protects the end user by, among other things, providing confidentiality of the user's data on the wire and authentication of the end-host to the user. It also provides a framework for authenticating the user to the end-host. In its default configuration, SSH requires a password on the server, but it can also be configured to use public keys.

To configure SSH for public keys, the user generates one or more key pairs (DSA or RSA) and distributes the public key(s) to the desired servers. The private keys are encrypted with passwords and stored on the local host. During the login process, the SSH client prompts the user for the password to his private key and uses it to decrypt the key. It then uses the decrypted key to generate a signature which is sent to the server. If the server can verify the signature, it allows the user to log in.

SSH provides an additional tool SSH-agent which assists in managing the private keys and further assists in multi-hop sessions. After it is started, the user registers each of his private keys and the SSH-agent takes responsibility for each. The agent prompts the user for the private-key passwords, then loads the keys into protected memory. The agent also creates a UNIX-domain socket on the local host, and stores its location in a well-known environment variable. The socket can be thought of as a tunnel directly to the agent, and all communication with the agent is through this socket. The SSH client then uses the tunnel to query the agent for authentication material during the login process.

SSH also provides an option to forward this socket to subsequent hosts. This allows SSH connections started multiple SSH hops away from the client's local host to connect back to the SSH-agent on the local host and authenticate using the material it manages.

One of the great strengths of SSH with public-key authentication is that a user can log in to an untrusted host without providing any sensitive data. The user provides only his public key and a signature. Even if the remote host is compromised, the user's authentication material is safe. Compare this to password authentication; if the remote SSH daemon has been compromised, an adversary can obtain the plaintext password. Any tool on the remote host which uses password authentication is susceptible to such an attack. By default, sudo is one of those tools.

The UNIX sudo command is designed to allow users to run commands as other users. It has a rich configuration language which allows the system administrator to delegate authority to execute commands as root (or other users) to particular users. It also provides detailed auditing records. The most common case is to use sudo for administrative purposes.

[1]This work was partially supported by NSF Grants CNS-04-26623 and CNS-07-14647. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the U.S. Government.

In this common case, the user instructs sudo to execute a command, and sudo checks a configuration file to verify that the user has permission to run the command as root. It then authenticates the user via password. If the password is correct, sudo then executes the command as the root user, while logging the details. In some environments [12, 5], sudo has supplanted the need for a root user entirely.

The sudo authentication mechanism can be configured to require the user's password (by default), the root password, or no password. The sudo package also includes extra authentication modules for Kerberos, Secureware, and SecurID, among others. The extra authentication modules operate in addition to the password requirements. That is, depending on the configuration, a user may have to enter his Kerberos password, and the root password. There is no module for public key-based authentication. Heavyweight schemes like Kerberos and SecurID require substantial investments in time and/or money, so the best options for small- to medium-sized networks are lightweight schemes such as password authentication or S/KEY [3].

S/KEY is a one-time password scheme which requires the end-user to maintain a hard-copy list of passwords, or to run a one-time password generator locally for each authentication attempt. Sudo requires re-authentication every five minutes, by default, so S/KEY is feasible for only the most dedicated. As a result, password authentication is the most common technique.

In this paper, we propose an extension to the sudo authentication mechanism to allow for public key authentication with the sudo engine. We describe our implementation of an BSD authentication module that uses SSH_AUTH_SOCK to authenticate incoming clients. We then describe the source code and configuration changes required to provide public-key authentication with sudo.

Related Work

The sudo [11] architecture was designed by Bob Coggeshall and Cliff Spencer at SUNY/Buffalo in the early 1980s for a VAX-11/750 running 4.1BSD. Control of sudo has passed through many hands in the intervening years. It is currently maintained by Todd Miller.

The SSH [15] protocol was designed in 1995 by Tatu Ylonen at Helsinki University of Technology. There are now several competing implementations of the original protocol and its derivatives, including the implementation released by SSH Communications Security (founded by Tatu Ylonen), and OpenSSH, developed by the OpenBSD project. Our work is based on the OpenSSH implementation.

The BSD Authentication framework [1] is an authentication framework used by some UNIX-like operating systems including OpenBSD and BSD/OS. It is similar in spirit to the Pluggable Authentication Modules (PAM) [10] found in Linux and Solaris. Neither provides a mechanism for interaction with the SSH authentication system.

Other related works include Kerberos [7] and LDAP [4] which provide unified network authentication mechanisms. LDAP also provides the Proxied Authorization Control [13]. In larger networks, RADIUS [8] and its successor DIAMETER [2] provide authentication, authorization and accounting protocols. They allow communication with a policy server to make policy-based decisions. These latter protocols are typically used for user administration in roaming and dial-up situations.

In [6], Napier describes a security flaw in the default sudo configuration wherein an attacker can obtain victim's sudo privileges without the victim's password. Sudo caches the user's password so that a user only has to enter his password every five minutes. That cache is valid for all TTYs on which the user is logged in. If the attacker can run an arbitrary process as the victim, then the attacker will have the same root privileges. Napier recommends turning off password caching entirely, but recognizes that this would encourage administrators to use a root shell to avoid retyping their passwords – a problem when it comes to auditing. Sudo with public-key authentication will allow the system administrator to turn off password caching, and does not require frequent re-typing of the administrator's passwords.

Sudo and SSH

Our implementation platform is the OpenBSD 4.2 operating system. This operating system ships with OpenSSH 4.7 and sudo 1.6.9p4, with OpenBSD customizations. In this section we will describe some relevant details from the BSD Authentication framework, sudo, SSH and the ssh-agent, as they exist in OpenBSD 4.2.

BSD Authentication Framework

OpenBSD uses the BSD Authentication framework, sometimes called bsd_auth, to present a uniform API to the various authentication modules. bsd_auth maintains a collection of modules in /usr/libexec/auth/. Each module performs a particular style of authentication, including password, Kerberos, S/KEY, and RADIUS, among others. A user program interacts with the authentication framework by making calls on the bsd_auth API. The API then executes the modules as separate processes in order to limit interactions between the child and parent processes, under the principal of least privilege [9].

The BSD Authentication framework is configured through the file /etc/login.conf. This file allows the user to add new authentication styles, and to assign styles to specific users or programs.

Sudo

The sudo command takes as a command line argument the command the user desires to have executed another user (typically root). It then searches for the user's login name in the configuration file /etc/sudoers, and verifies that the user has the correct permissions. Beyond this point, the sudo implementation in OpenBSD diverges slightly from the general sudo release.

The general sudo release is configured to support multiple authentication styles, including passwords, Kerberos, and SecurID. The code for implementing each of these styles is included with the sudo release. In OpenBSD, these are bypassed and the operating system's BSD Authentication framework is called instead. To effect this, the sudo authentication modules have been replaced with a single module bsdauth which, in turn, provides all interaction with the BSD Authentication framework. Thus, on OpenBSD, the authentication styles presented by sudo are those supported by the BSD Authentication framework.

Thus, the complete authentication process with sudo is as follows. The bsdauth module uses the BSD Authentication framework API to authenticate the user. The API uses the definitions in login.conf to determine the particular authentication type, and then loads the appropriate module from /usr/libexec/auth. The module is loaded, it performs the authentication process, and then completes with success or failure. If successful, sudo executes the given command in a cleansed environment. That is, all environment variables except LOGNAME, SHELL, USER, and USERNAME are removed.

SSH and the SSH Agent

The SSH-agent is used to facilitate the use of public keys with SSH. We will walk through a sample SSH connection, from an SSH client to an SSH server, with SSH-agent enabled on the client, to illustrate their interaction in the authentication process. Before the authentication process can begin, the keys must be generated using the ssh-keygen utility and the public key must be stored on the server – appended to the user's .ssh/authorized_keys file.

The user then starts the SSH-agent, and verifies that the SSH_AUTH_SOCK environment variable has been exported to the environment that will use the agent. The user adds his keys to the agent using the ssh-add command. The agent prompts the user for the password to each key and then loads them into memory for future use. Once the key has been added to the agent, the client is ready to initiate an SSH connection.

The SSH protocol architecture consists of a transport layer protocol, a user authentication protocol, and a connection protocol. The transport layer protocol provides encryption, integrity protection and server authentication. The transport layer is negotiated first. When it is complete, the user authentication protocol begins, as described in [14]. The client requests public key authentication:

```
byte       SSH_MSG_USERAUTH_REQUEST
string     user name
string     service name in US-ASCII
string     "publickey"
boolean    FALSE
string     public key algorithm name
string     public key blob
```

Where the 'public key blob' may contain certificates. If the public key matches a public key stored on the server, the server accepts the request:

```
byte       SSH_MSG_USERAUTH_PK_OK
string     public key algorithm
string     public key blob
```

The client uses the SSH_AUTH_SOCK tunnel to obtain from the agent a signature over the following data:

```
string     session identifier
byte       SSH_MSG_USERAUTH_REQUEST
string     user name
string     service name
string     "publickey"
boolean    TRUE
string     public key algorithm
string     public key for authentication
```

And returns it to the server:

```
byte       SSH_MSG_USERAUTH_REQUEST
string     user name
string     service name
string     "publickey"
boolean    TRUE
string     public key algorithm name
string     public key for authentication
string     signature
```

If the server is able to verify the signature, the authentication has succeeded.

The agent forwarding option on SSH maintains the SSH_AUTH_SOCK variable at each hop in a multi-stage SSH connection. This means that when a client connects to server h_1, the SSH_AUTH_SOCK environment variable is re-created there, tunneled back to the SSH-agent on the client machine. When the user then initiates a connection from h_1 to h_2, the SSH process uses the same SSH-agent connection, with the same authentication process described above, connecting through the SSH_AUTH_SOCK on h_1. We take advantage of the fact that this connection exists and use it to leverage the authentication process to create public-key sudo.

Implementation

Our implementation consists of the addition of a new authentication style login_pubkey to the BSD Authentication framework, and re-configuration of sudo to make use of the new style. We link the login_pubkey module to libssh during compilation, which allows our module to manipulate SSH keys, request signatures, and call other functions used by the SSH client and SSH server.

We modify the login.conf file to add login_pubkey to the BSD Authentication framework authentication styles. We also modify the sudoers file, using the env_keep directive, to preserve the SSH_AUTH_SOCK environment variable for the authentication module.

When an authentication module is invoked by the BSD Authentication API, it receive a number of arguments from the calling process. These arguments provide the necessary information for authentication to occur. Parameters include the name of the user being authenticated, the authentication service being requested, and several other options, including details on whether the authentication service being requested is a challenge or a response. Since the challenge/response portion of the SSH authentication are handled outside of sudo, the challenge service is ignored by login_pubkey; all work is performed during the response service.

When login_pubkey receives a service request of type *response*, this indicates that the parent authentication process is awaiting an authentication decision. The module loads the SSH_AUTH_SOCK environment variable and opens the socket which tunnels to the SSH-agent, using the SSH client function ssh_get_authentication_connection(). The module then queries the agent for key details using the ssh_get_first_identity() and ssh_get_next_identity() functions.

With the private key details in hand, login_pubkey then searches the user's authorized_keys file for a corresponding key by calling the SSH server function user_key_allowed(). It requests the agent signature by calling ssh_agent_sign, and verifies it by calling key_verify(). If it succeeds, then user has successfully authenticated. If any of the functions above fail, the process is considered a failed authentication.

This process is identical to the authentication process performed in SSH, with login_pubkey serving as both SSH client and SSH server with respect to the SSH-agent. At process completion, a standard SSH public key authentication has taken place, and no sensitive information has been revealed on the server. Furthermore, with agent forwarding, the SSH_AUTH_SOCK is recreated, tunneling back to the original agent, on each subsequent hop in a multi-hop session, so this process remains unchanged even in multi-hop sessions.

Example Session

In this section, we will walk through an example session using sudo with public key authentication. The user starts his SSH-agent, and checks to make sure that the SSH_AUTH_SOCK environment variable has been created. This variable contains the filename of the UNIX domain socket connected to the agent.

```
castor% ssh-agent
castor% echo $SSH_AUTH_SOCK
/tmp/ssh-aHYaC22922/agent.22922
```

The user then adds his private key to the agent. The agent now manages the private key, and it is now possible to make queries through SSH_AUTH_SOCK against this key. We also make use of the -c option which is discussed in the following section.

```
castor% ssh-add -c .ssh/id_rsa
Identity added: .ssh/id_rsa
```

Next, the user connects to the remote server making sure to enable agent forwarding. After connecting, the user checks to make certain that the SSH_AUTH_SOCK has been forwarded.

```
castor% ssh -A pollux
pollux% echo $SSH_AUTH_SOCK
/tmp/ssh-kOfZrk1118/agent.1118
```

It is now possible to use sudo with public key authentication:

```
pollux% ./sudo -a pubkey /bin/ls
file.1  file.2  hello.txt
pollux%
```

From the server pollux, connect to another server clytemnestra, again with agent forwarding.

```
pollux% ssh -1 -A clytemnestra
clytemnestra% echo $SSH_AUTH_SOCK
/tmp/ssh-jjneu20310/agent.20310
```

Sudo on clytemnestra uses the local SSH_AUTH_SOCK which tunnels through pollux to the agent on castor and authenticates the user.

```
clytemnestra% ./sudo -a pubkey /bin/ls -a
this_is_an_empty_file  hello.txt
clytemnestra%
```

As long as the SSH_AUTH_SOCK is forwarded, the number of intervening hops does not affect the authentication mechanism.

Discussion

The adversary we consider is one who has a root compromise on the remote host. He may have inserted, among other things, malicious replacements for the SSH and Sudo executables and the authentication modules. Our goal is to prevent the adversary from obtaining any sensitive authentication materiel (such as a password).

From this adversary, SudoPK can be viewed as an API on top of SSH agent forwarding. It does not provide any additional functionality, it simply provides easier access to the agent. Hence, we must focus on attacks on the agent itself.

An adversary who has access to a user's SSH_AAUTH_SOCK can use it as an oracle to generate signatures for connecting to any host for which that user has permission. SSH-agent provides protection against this attack by allowing the user to add keys with the -c option. This option requires the local-host identity be confirmed (via password) before every signature. In this fashion, the end-user will be notified of unauthorized authentication attempts using the SSH-agent.

Even with the -c option in place, the adversary may attempt to race for the SSH_AUTH_SOCK during a valid attempt. If the adversary wins, the end-user will receive the -c password prompt as expected. To prevent this attack, which is an attack on agent forwarding, regardless of whether SudoPK is in place, we modify the SSH agent to present the user with the data to be signed and requesting confirmation before generating a signature.

Conclusion

The public-key sudo mechanism is an implementation that solves a specific and common problem. However, the concepts used here are generic and can be extended. The notion of public-key authentication through secure tunnels, as implemented in SSH with agent forwarding, is quite powerful. The login_pubkey module is a generic interface to that mechanism and can be used by any application, not just sudo. We believe that this architecture is applicable in other scenarios, including remote attestation. In this scenario, the trusted platform module (TPM) takes the place of the SSH-agent, and queries against the TPM may be made through the corresponding agent tunnel.

Author Biographies

Mack Lu received his B.S. in Computer Engineering from Columbia University. During his undergraduate years he worked at the Network Security Laboratory. He is currently at Google.

Matthew Burnside is a Ph.D. student in the Department of Computer Science at Columbia University. He works for Professor Angelos Keromytis in the Network Security Lab. He received his B.A in Computer Science and M.Eng in Computer Science and Engineering from MIT. His research interests are in network anonymity, trust management, and enterprise-scale policy enforcement.

Angelos Keromytis is an Associate Professor with the Department of Computer Science at Columbia University, and director of the Network Security Laboratory. He received his B.Sc. in Computer Science from the University of Crete, Greece, and his M.Sc. and Ph.D. from the Computer and Information Science (CIS) Department, University of Pennsylvania. He is the author and co-author of more than 130 papers on refereed conferences and journals, and has served on over 70 conference program committees. He is an associate editor of the ACM Transactions on Information and Systems Security (TISSEC). He recently co-authored a book on using graphics cards for security, and is a co-founder of StackSafe Inc. His current research interests revolve around systems and network security, and cryptography.

Bibliography

[1] *BSD Authentication System*, http://www.openbsd.org/cgi-bin/man.cgi?query=bsd_auth .

[2] Calhoun, P., A. Rubens, H. Akhtar, and E. Guttman, "DIAMETER Base Protocol," *Internet Draft*, Work in progress, Internet Engineering Task Force, December, 1999.

[3] Haller, Neil M., "The S/KEY One-time Password System," *Proceedings of the Internet Society Symposium on Network and Distributed Systems*, pp. 151-157, 1994.

[4] Harrison, R., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms," *RFC 4513*, June, 2006.

[5] *Mac OS X*, http://www.apple.com/macosx .

[6] Napier, Robert A., "Secure Automation: Achieving Least Privilege with SSH, Sudo and Setuid," *18th Large Installation System Administration Conference*, pp. 203-212, November, 2004.

[7] Neuman, B. Clifford and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications*, Vol. 32, Num. 9, pp. 33-38, 1994.

[8] Rigney, C., A. Rubens, W. Simpson, and S. Willens, "Remote Authentication Dial In User Service (RADIUS)," *Request for Comments (Proposed Standard) 2138*, Internet Engineering Task Force, April, 1997.

[9] Saltzer, Jerome H. and Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, Vol. 63, Num. 9, pp. 1278-1308, 1975.

[10] Samar, V. and R. Schemers, *Unified Login with Pluggable Authentication Modules (PAM)*.

[11] *Sudo*, http://www.sudo.ws .

[12] *Ubuntu 8.04*, http://www.ubuntu.com .

[13] Weltman, R., "Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control," *RFC 4370*, February, 2006.

[14] Ylonen, T., "The Secure Shell (SSH) Authentication Protocol," *RFC 4252*, January, 2006.

[15] Ylonen, T., "The Secure Shell (SSH) Protocol Architecture," *RFC 4251*, January, 2006.

STORM: Simple Tool for Resource Management

Mark Dehus and Dirk Grunwald – University of Colorado, Boulder

ABSTRACT

Virtualization has recently become very popular in the area of system engineering and administration. This is primarily due to its benefits, such as: longer uptimes, better hardware utilization, and greater reliability. These benefits can reduce physical infrastructure, space, power consumption, and management costs. However, managing a virtualized environment to gain those benefits is difficult and rife with details.

Through the use of a concept known as virtual appliances, the benefits of virtualization can be brought to organizations without sufficient knowledge or staff to install and support a complex virtual infrastructure. This same concept can also be used to provide cheap datacenter services to larger companies, or research facilities that are unable or unwilling to run a high performance computing environment.

In this paper, we describe Storm, a system designed to simplify the development, deployment and provisioning for common applications. The system is designed to be easy to configure and maintain. It can automatically react to changes in system load to deploy additional services and it dynamically powers client machines using IMPI controls to enhance energy savings. We demonstrate the utility of the system using scalable mail appliance.

Introduction

Virtualization has become very popular as a way of managing a large number of complex software systems. This is primarily due to its benefits, such as longer uptimes, better hardware utilization, and greater reliability enabled by the ability to move a virtual machine from one host computer to another. These benefits lead to reduced physical infrastructure/footprint, power consumption, and total cost of ownership [1].

However, managing virtual environments is complex; a number of management frameworks, both commercial and academic or open source projects, have been recently developed. These frameworks seek to reduce the complexity of managing a large scale deployment or infrastructure. Usually, these frameworks are complex – that complexity is introduced in a large part by their attempted generality. We argue that we can produce a simpler tool by taking a restricted view of how many information technology organizations actually conduct their operations.

We argue that the concept of *layered virtual appliances* should be central to the development and deployment of a virtual machine management framework – so much so that we are focused on a *virtual appliance management framework* rather than a *virtual machine management framework*. By adopting this focus, and using a simple, extensible framework for managing such appliances, we show how virtualization can be brought to organizations without sufficient knowledge or staff to install and support a complex virtual infrastructure. We also explore how virtualization can be used to provide cheap datacenter services to larger companies, or research facilities that are unable or unwilling to run a commercial management framework. In each case, we're focused on a simple management framework that is easy to adopt.

Server sprawl and operating system (OS) management are major concerns in the area of information technology [2]. This paper addresses these concerns by simplifying the use of virtualization and system configuration for application developers and system administrators. We also show how our simplified interface can still be used to provide scalable "on demand" computing services using standard interfaces and technologies.

As we describe in more detail later, virtual appliances [3], are a combination of operating system and application components that provide a specific computing task, such as spam filtering, mail delivery or web serving. The STORM system provides a *virtual appliance configuration and provisioning system*. The STORM management node controls a cluster of computers that use a virtualization hyperviser, such as Xen, VMWare or VirtualBox. Each node in the cluster must run a specific control program (not shown) that coordinates the STORM management node. Administrators (or programs) can cause virtual appliances to be deployed on nodes within the cluster of computers. The STORM system determines on which nodes the appliance should be run, loads the appropriate configurations and customizes them for the environment. Given system management mechanisms such as IPMI, the STORM system can also dynamically manage the power state of different computing nodes to reduce the energy needs based on load and configuration. Using monitoring interfaces provided by common hypervisors, the STORM system can cause new appliance instances to be generated when CPU utilization or reported host demands warrant.

As we'll describe in related work section, there are many existing virtual machine management systems. Some of these are designed for specific applications, such as managing "grid" computing or clusters of machines. The approach we took in the STORM system is to focus on simplicity and ease of infrastructure maintenance. The framework is simple because it uses readily available technologies (reducing the time for installation) and presents a simple but very capable web interface for system management. To simplify ongoing management and deployment of applications, each deployed application contains four "layers":

1. A *common operating system substrate* that contains the basic components needed by all virtual appliances; the Ubuntu "Just Enough OS" (JEOS) platform is a representative example of this substrate.
2. An *appliance specific component* that provides the application and necessary libraries; an example might be the the "postfix" program, ldap and mysql libraries for remote mail deliver and other necessary libraries.
3. A *deployment specific component* that customizes the combination of the operating system substrate and the appliance specific component; an example might be the configuration files for postfix, mysql, nfs and ldap. The deployment specific component essentially captures *changes* to the underlying appliance component – for example, the appliance component would typically include "off the shelf" configurations provided by an O/S distribution. The deployment specific component would be the result of an appliance maintainer editing the specific configuration files to customize those files for the local environment.
4. An *instance specific component* that uses information provided by the STORM server to configure a specific instance of a more general appliance. For exmaple, that instance specific information may configure the domain name to be "mail.foo.com" *vs.* "mail.bar.com" and makes (minor) changes to /etc/sendmail configuration files based on data from a specific configuration file accessed by the STORM server.

On Linux, the STORM system can deploy an "initial RAM disk" that combines these four layers using the "union file system." This configuration allows a single O/S configuration to be used by multiple clients, and that single configuration can be provided by NFS or iSCSI. Using the union file system in such a structured fashion greatly simplifies the task of building the "deployment specific component" – the local administrator essentially logs in to an instance of the machine and updates the configuration files. The top "writable" layer of the union filesystem will capture those changes. Many of the changes will involve adding instance specific components, which are specified as "variables" in the configuration files that are later expanded when the instance is created.

These layers allow the base O/S and the specific applications to be split; this means that an underlying O/S image, including all the common libraries and management tools, can be patched without having to then patch each individual appliance component. Reducing the number of operating system configurations greatly reduces the security risks of multiple unpatched configurations; it also reduces the workload on the administrator. Likewise, the separation of the appliance specific component, the deployment specific component and the instance specific allows the common components to be upgraded without having to reconfigure each component; again, this assists in securing those appliance components. It also reduces the storage needed for the underlying O/S substrate – while this seems like a minor feature, having many appliances use the same O/S image means that the file server can more effectively cache common utilities, speading access and reducing needed resources.

This layering is not perfect – depending on the package management system used by the underlying O/S, it may be that an appliance component may install patches already provided by an updated or modified O/S substrate. In our experience, systems similar the "debian" package management system provide the most flexible interface – these systems stores package information in individual files, rather than in databases as is done in the common "redhat" package management system. Either packaging system works, but the database versions will consume more space.

These individual components can be automatically combined when configuring the provisioned system. STORM can also be used to exert finer levels of control should the "virtual appliance" model not be sufficient; however, these are not the focus on STORM nor of this paper. Likewise, the STORM system also provides a secure XML-RPC service that can be used by appliances themselves to control provisioning based on environmental or load conditions.

Again, by offering guidence in how a system should be configured an by limiting the scope of problems that we try to address, we believe that STORM provides a simplified workflow for an IT administrator. In this remainder of this paper, we briefly describe the virtualization technology that underpins STORM. Then, in the Method section we describe the components that make up STORM. In the Example and

Analysis section, we walk through an example of using STORM to configure an energy and load-aware mail processing system. We subject that system to artificial load and demonstrate that STORM is capable of adaptively adjusting the number of mail processing appliances. Lastly, in section Future Work, we describe similar systems and future directions for STORM.

Background

Virtualization is not a very new area of computer science. The concept of virtual machines date back to the 60's with the IBM research system titled CP-40 [4]. This system is one of the first known to be able to run multiple instances of client operating systems within it. Virtualization gained popularity as a management tool with the development and widespread deployment of VMware circa 1998. The VMware virtualization technology took a different approach than the IBM hardware – the virtualization was done by binary rewriting.

In 2003, Xen, developed at the University of Cambridge [1], introduced a *para-virtualization* system, where in the "guest" operating system cooperated with the virtualization system to reduce virtualization overhead. Their described an implementation of a hypervisor that made para-virtualization possible on the x86 architecture. The code for this implementation was released under an open source license, and distributed freely on the Internet, greatly increasing the popularity of virtualization on commodity platforms. Since that time, Intel and AMD have provided additional hardware support to improve virtualization performance.

With the availability of an inexpensive and high performance virtualization system, many projects have been started using this technology to provide homogeneous computing, in which the operating system is independent from the hardware.

The *hypervisor* is the software that enables multiple operating systems to run on a single physical host. It is the intermediary between the operating system being virtualized and the physical host. The hypervisor is also responsible for handling time sharing between virtual machines. There are several hypervisors currently available, a few worthwhile mentioning are: Vmware, Xen, KVM, and Virtual Box. In this paper, our description of the STORM system is based on the Xen hypervisor, but STORM is not limited to that virtualization hypervisor. STORM uses the libvirt virtualization library to interface to the hypervisor, meaning it can support Xen [1], Qemu [5], KVM and "container systems" [6] such as the Linux Container System and OpenVZ. The libvirt interface can be easily extended to other virtualization systems.

A *virtual machine* is the guest operating system being controlled by the hypervisor. It contains the application(s) that a specific user desires to run. Depending on the type of virtualization used, the operating system can be run on the hypervisor without any modification.

When using Xen, a virtual machine is typically referred to as Domain*U*, where *U* is a unique number to the specific virtual machine. The number 0 is reserved for a special domain that has escalated privileges for management purposes.

A *virtual appliance* [7, 8] is the definition of a virtual machine designed to performance a specific application. The definition typically includes metadata describing information about services provided, resources required, and dependencies. The metadata is typically stored in a portable format, such as XML.

There are two types of ways to describe a virtual machine. One method [7, 8, 9, 10] describes it entirely in metadata. All information regarding packages to install, ports to open, services to configure, is defined in metadata. The software creating virtual machines from the definition will take this description and do everything necessary to make sure the virtual machine created is exactly as the author intended. Configuring the system solely from metadata is very extensible; however it requires more work from appliance developers. There are tools available that can reduce this workload.

Another method takes the combination of metadata, and a disk image containing a pre-configured version of the operating system and all software desired. This method is not as extensible as the first since it requires the distribution of a hard disk image. However, it allows the developer to have more freedom in configuring the virtual appliance and requires adoption of fewer tools. This is the approach we have take in STORM, but we've extended the basic "disk image" approach by using *layered* appliance deployments. We describe this method in the next section.

Method

The overall STORM system consists of three primary entities:

1. the STORM manager (see the Virtual Appliance Server section)
2. channel server (see the Channel Server section),
3. a disk image server (that may be integrated with the channel server),
4. and the virtual appliance server (see the Appliance Definition section).

The interactions between these entities are shown Figure 1. The management appliance plays the most important role in the system. It is responsible for creating & controlling virtual machines, appliance servers, and for fetching appliances. In practice, the STORM manager is a virtual appliance. Having the management software implemented inside a virtual appliance provides increased security, simplified installation, the ability to run on any available virtual appliance server, and adheres to the guidelines given by the Xen creators [1].

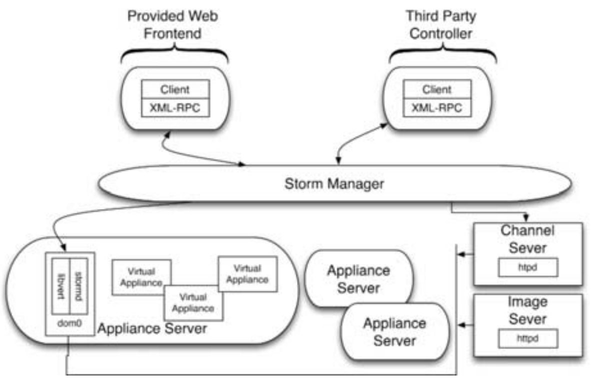


Figure 1: STORM system components and interactions.

STORM provides an easy-to-use web interface which is programmed in Python with the help of a framework called TurboGears [11]. This web interface gives administrators the ability to control the current state (running, stopped, paused) of any virtual machine, install new appliances, and manage available appliance channels.

Each appliance in the STORM cloud receives a DHCP address from the STORM appliance, or if configured, from an external DHCP server. This address can be dynamic or statically configured. Running a DHCP server on the management appliance prevents other appliance developers from having to worry about network configuration.

The STORM appliance also provides Kerberos and LDAP services. This allows for the customer to have a centralized user and password database against which virtual machines can authenticate. Again, our emphasis on simplicity of system configuration and common IT tasks led us to provide such a centeralized authentication and authorization service. That service also gives granular control over which virtual machines users have access to. For example; Bob can be detailed to have access to upload files to the web server but not make alterations to the MySQL database. The services provided are similar to Microsoft Active Directory, but use open source software and database schemas. If desired an external or appliance based active directory server can be used instead.

Lastly, the STORM appliance can control the power state of the client machines using the Intelligent Platform Management Interface (IPMI) [12]. Client machine hardware requirements depend on the number of virtual machines desired to be ran concurrently. The amount of RAM should always be 128 MB greater then the amount required to run the desired virtual machines. This ensures that domain 0 has enough memory available to operate the STORM control daemon. There should also be sufficient cores available to meet the requirements of each virtual machine.

Virtual Appliance Server

In order to accomplish these tasks the management software communicates with two daemons running within Domain0 on any given virtual appliance server. One of the daemons is libvirtd, which provides remote access to the Xen API [13]. The other (stormd) is specifically designed for this project and provides access to services and information that cannot currently be obtained from the virtual appliance server through libvirtd. These services include: appliance retrieval, upgrades, virtual machine creation, removal, virtual appliance IP address reassignment, shutdown and restart. The daemon is a XML-RPC server implemented in python and it extends the built-in XML-RPC server class. It listens for and handles connections from authorized STORM virtual machines. It is the sole piece of software responsible for handling the services described above.

Both of these daemons authenticate and encrypt communications from the STORM virtual machine using SSL. Each virtual appliance server has a local certificate authority that is responsible for generating, signing and providing a public/private key pair to the management appliance. The appliance then uses that key pair when connecting to either daemon when requesting to perform tasks.

Channel Server

A channel server is responsible for distributing and providing virtual appliances to STORM users. The user points the STORM appliance server to the desired channel, and then all information regarding appliances available is cached. After this step is completed, the user may create virtual machines based off the appliances on the channel server.

The information about available appliances is stored in an RSS feed, which are currently unlocked due to the scope of the project. The RSS feed provides URL's to the location of each appliance, allowing for load to be distributed across multiple servers. The feed and appliances are accessed via the HTTP protocol. Any standard HTTP server is suitable to act as a channel server, however Lighttpd is recommended. Lighttpd offers several performance enhancements over others and also requires very few resources to run.

Appliance Definition

The metadata that goes with a given STORM appliance is defined entirely in XML. This allows other systems to easily recognize and parse an appliance. It also makes it much simpler for a developer to create an appliance as they do not have to obtain knowledge of a proprietary format.

A key difference between the STORM appliance definition and other appliance definitions is simplicity. Formats such as the Open Virtual Machine [14], or CVL [8] are either too complex or non-trivial to parse. The STORM appliance definition contains 'bare minimum' metadata to describe a virtual appliance. The remaining information required is stored within the appliance image itself. Some sample fields of the data supplied are: label, description, resource requirements, dependencies, and control panel URL. A sample configuration is shown in Figure 2 A full description of the specification can be found in Appendix A.

API Implementation

Upon connection to either the libvirt or storm daemon, the management appliances verifies the authenticity of the daemon it is connecting to, and the daemon verifies the authenticity of the management appliance. Once the connection has been established, the management appliance may issue numerous procedure calls. The following highlights procedure calls that are important to the opperation of the system as a whole:

- get_downloadprogress(): Returns the progress of a current operating system, or appliance in a percentage number less than 1. This function will return -1 if nothing is currently being downloaded.
- get_downloadspeed(): Returns the speed (in kilobytes per second) at which a current operating system or appliance is downloading at. Returns -1 if nothing is currently being downloaded.
- get_diskspace_used(): Gives amount of physical disk space allocated in megabytes for virtual machine usage. Actual usage information is available only from the virtual machine itself, and is currently the responsibility of the appliance developer to give per virtual machine disk usage. Future work includes developing a method for obtaining per virtual machine disk usage.
- get_virt_uptime(virtual_machine_id): Provides uptime information for a given virtual machine.
- server_info(): Returns the version of the server, list of capabilities, and other general information such as uptime.
- server_upgrade(): Updates all programs running on the virtual appliance server.
- server_set_network(self,...): Sets the network configuration for the virtual appliance server identified by the specific network configuration (IP address, netmask, gateway, primary and secondary DNS server). If the DHCP flag is set then all other network parameters are ignored and information is taken from DHCP.
- disk_create(): Create a blank or empty disk for the instance about to be deployed; this image is used to store automatically modified configuration files and data files for the instance.
- app_install: Downloads an appliance image (e.g., a sendmail appliance) from the channel server; instances can be created from this image.
- app_check: Checks to see if an appliance image exists (if it does not, the image is retrieved using app_install).
- app_upgrade: Forces an upgrade for an appliance, downloading a fresh image from the channel server and replacing the current appliance image with that updated image.
- os_install: This is similar to the app_install method, but it specific to the operating system layer rather than the application layer.
- Additional interfaces: There are a number of interfaces that provide services complimentary to the ones described above. We omit the detailed description for these interfaces: server_reboot(), server_shutdown(), disk_remove, app_remove, os_check, os_upgrade, and os_ remove.

Scalability

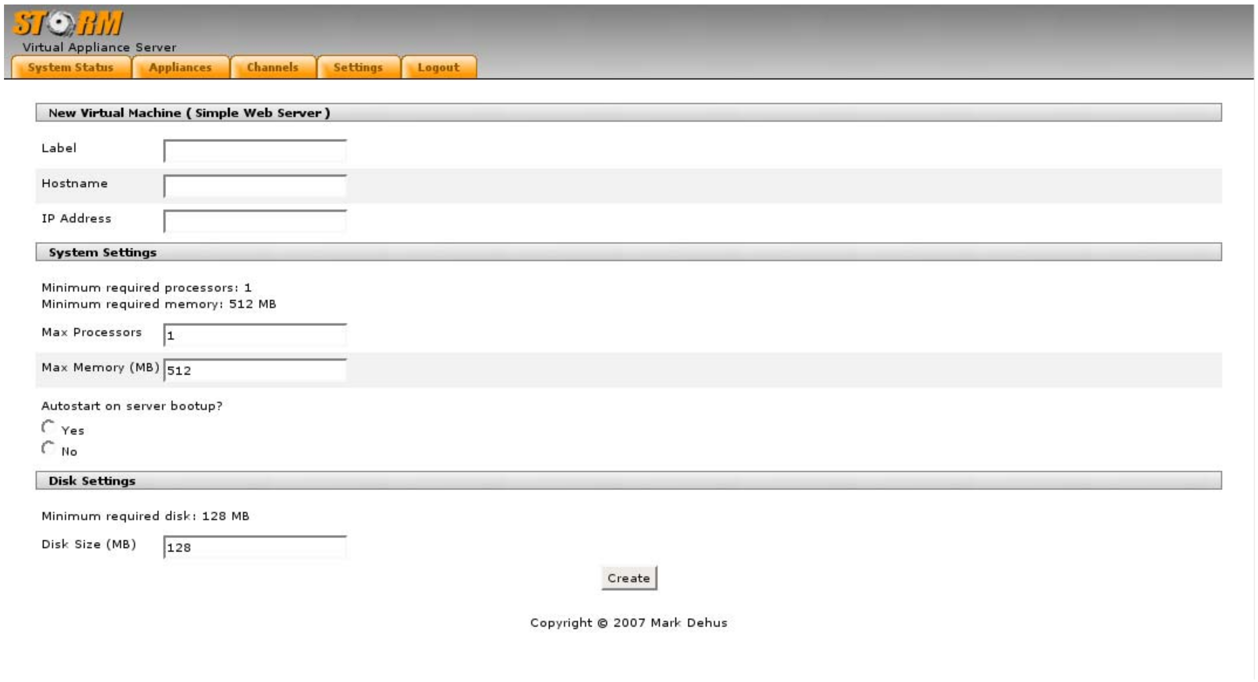
With the addition of network attached storage, or a storage area network the STORM system will scale to support a very large number physical hosts and virtual machines. It can theoretically address up to 2^{32} (size

```
<?xml version='1.0' encoding='utf-8'?>
<appliance version='1.0'>
    <label>Simple Web Server</label>
    <description>A very simple and efficient test web server.</description>
    <version>1.0.0-0</version>
    <size>1024</size>
    <url>http://cs.colorado.edu/appliances/webserv.tgz</url>
    <sig>http://cs.colorado.edu/appliances/webserv.asc</sig>
    <provides>webserv</provides>
    <webpanel>/upload.php</webpanel>
    <hardware>
        <cpus>1</cpus>
        <memory>524288</memory>
        <ostype>linux</ostype>
        <disk name="hda1" type="system" file="sys.img" />
        <disk name="hda3" type="swap" size="128" />
        <disk name="hda2" type="user" size="128" />
    </hardware>
    <dependency>emailserv:1.0.0-0</dependency>
    <dependency>firewall</dependency>
</appliance>
```

Figure 2: Sample appliance configuration.

of an unsigned integer) physical hosts on a 32-bit architecture.

The channel server can also be scaled to meet demand from multiple clients. This can be done by having a single server maintain the RSS feed, and an index of appliances available. The actual appliances themselves can be stored across multiple image servers with different URL's or on a single URL with an HTTP load balancer to dynamically redirect traffic to servers which are least busy.



New Virtual Machine (Simple Web Server)

Label	
Hostname	
IP Address	

System Settings

Minimum required processors: 1
Minimum required memory: 512 MB

Max Processors	1
Max Memory (MB)	512

Autostart on server bootup?
Yes
No

Disk Settings

Minimum required disk: 128 MB

Disk Size (MB)	128

Copyright © 2007 Mark Dehus

Figure 3: Web panel used to create virtual appliance.

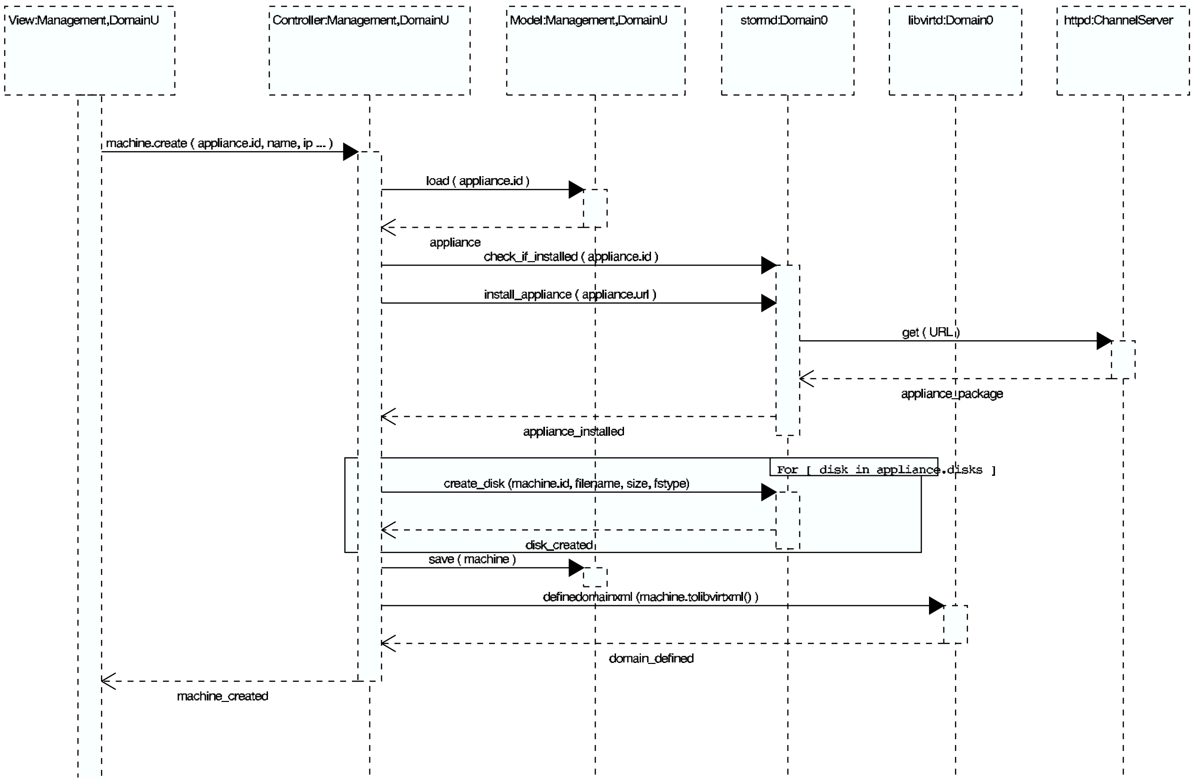

View:Management,DomainU
Controller:Management,DomainU
Model:Management,DomainU
stormd:Domain0
libvirtd:Domain0
httpd:ChannelServer
machine.create (appliance.id, name, ip ...)
load (appliance.id)
appliance
check_if_installed (appliance.id)
install_appliance (appliance.url)
get (URL)
appliance_package
appliance_installed
For [disk in appliance.disks]
create_disk (machine.id, filename, size, fstype)
disk_created
save (machine)
definedomainxml (machine.tolibvirtxml())
domain_defined
machine_created

Figure 4: UML description of system component interactions.

Both the physical hosts and channel servers can easily scale without any problems and the only potential bottleneck is within the management software. Any performance issues in the software can simply be resolved by fixing the problem area within our code. We did not have sufficient hardware to test a large scale installation, but did not encounter any problems on the setup with our 16 node cluster.

Usability

The web interface for the STORM system was designed with Palmer's five constructs [15] for website usability in mind. The interface was also designed to meet the following standards: XHTML4, CSS1, and 508. Each of these standards helps to ensure that the interface will be portable, readable, and easily accessible in all browsers.

Security

In order for the STORM system to be secure two important areas must be resistant to attacks: communications and software. Secure communications are required to ensure that an attacker cannot listen in or hijack any connections between a user and the web interface, or between the web interface and management daemons running on the hypervisor. Secure software is required to prevent an attacker from exploiting bugs in code to gain unauthorized access, or to make the software behave in an undesired manner.

Secured communications are achieved through the use of TLSv1, which at the time of writing is a known to be secure protocol [16, 17]. Each session is authenticated using SHA1 signed certificates, and encrypted under AES128. This includes: the session between the user and the web interface, and between the web interface and hypervisor. For the purpose of testing self-signed certificates were used, however in a production environment all certificates would be signed by a commonly known certificate authority such as VeriSign.

The primary area of concern with software security is in the code that provides network functionality. Standard python libraries were used to provide networking functionality within the storm daemon. These libraries are xmlrpclib [18] and m2crypto [19]. They are both open source, community maintained projects. No custom code was written to provide network functionality; therefore the storm daemon is secure as long as the libraries used are also secure. At the time of writing there are no known exploits for the versions used in either library. Furthermore, once libvirt receives more development in the area of virtual appliance management it could eliminate the requirement for the storm daemon altogether.

Component Interaction

The interaction of the components can best be explained by describing the interactions during specific operations, such as virtual machine creation and configuration, as shown in Figures 3 and 4.

In that example, the Web interface (Figure 3) is used to create a specific machine instance. The UML descripton, shown in Figure 4, indicates that request is relayed to the STORM controller written in Python, running on the management appliance; the controller then communicates with the STORM daemon using XML-RPC to determine if the appliance is already available on that specific system; if not, the appliance is installed by retrieving the required disk image from the channel server. The STORM daemon then contacts the STORM management controller to inform it that the appliance has been installed; the controller then directs the storm daemon to create disks and other resources as directed by the appliance specification. Once those resources are created, the controller defines a new Xen domain, configures the network resources indicates to the the user that the machine has been created. Note that the entire process can be controlled by XML-RPC and does not need to be driven by the web interface.

Example and Analysis

To illustrate the capabilities of the STORM system, we use the XML-RPC interface and the Xen kernel monitoring utilities to implement a scalable email processing system. In this scenario, we configured an email appliance using the ''postfix'' MTU. To demonstrate the power control and automatic provisioning made possible by STORM, we used two slots or blades in a Dell M1000 cluster with M605 blades (dual socket, four cores, 3.0 GHz, 16 GB RAM) as the mail processing engines. We used an additional four blades as load generators to subject the mail processing engines to extreme load. The Dell M605 blades provide an IPMI interface that allows us to measure power usage (in Watts) as well as control the power state of individual blades. We used information from the Xen virtual machines to estimate server loads; an alternate mechanism would be to monitor SNMP data from individual operating systems, but we focused on O/S-independent mechanisms and mechanisms that would be available even if the guest O/S is subject to intense service loads. We determined that a guest O/S is overwhelmed when the assigned CPU utilization is at 90%, and that a physical host is overwhelmed when the overall CPU utilization of the host is at 90%. The CPU utilization is based off the number of CPU seconds used as provided by Xen.

In the original configuration, we deployed a single mail processing utility; each mail agent simply discarded email's that were successfully delivered. Following the start of the mail processing system, we enabled the load generating programs (which were also configured as STORM appliances on alternate blades); those programs produced 25 MB mail messages at a rate of 2000 per second.

CPU Frequency scaling was not deployed on the test system, thus the increases in power usage as appliances are being brought online is not very visible until a new node is turned on. Power fluctuation is mostly attributed to the boot process of the second node. During POST all fans are spun at full speed, disks are spun up, and initialization procedures are run for the underlying hypervisor and Domain.

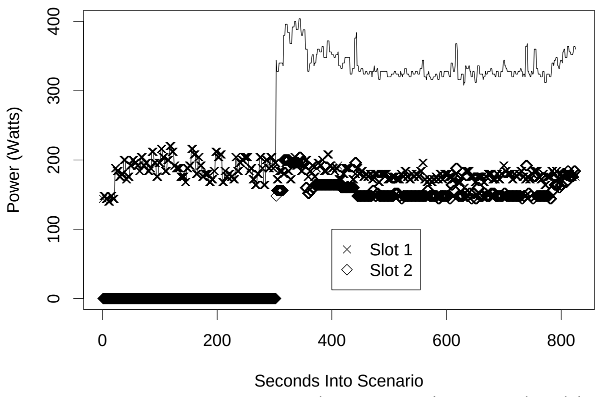


Figure 5: Power Usage Under Increasing Load. This diagram shows measurements of instantaneous power usage collected using an IPMI interface as two blade slots are used for mail processing. The individual data points indicate the power for the individual blades and the line indicates the power for the combined set of blades.

Figure 5 shows a time series plot of the power consumed by the individual mail processing systems. At approximately 300 seconds into the experiment (shortly after the load generating programs were enabled), the reported load for "Slot 1" (the primary mail processing program) was sufficiently large that the STORM monitoring component elected to configure a second mail processing node. The system was configured and deployed using the mechanisms described earlier. The mail processing systems have multiple MX and A associated records. Every time Storm brings up a new instance of the mail appliance, the necessary records are automatically added to the domain if STORM is designated as the primary DHCP & DNS server. This effectively balances the load across all running mail servers. As you can see in Figure 5, while the demand is low, total system power is low because one of the processing nodes is shut off. As the demand rises, more instances of the mail appliance are created on the first eight-core processing node, causing an increase in power. Eventually the first appliance server starts to reach maximum capacity, causing the second eight-core blade slot to be turned on. When the second appliance server becomes available then new instances are created on whichever blade slot has the lowest load. Figure 5 shows the second processor ("Slot 2") being enabled at about 300 seconds. There is a short burst of maximum power as the system undergoes self-test and then individual cores are allocated for for processing tasks.

This entire process is not automatic – in particular, the configuration of our round-robin DNS server is an afterthought and somewhat grafted to the other infrastructure. However, using the XML-RPC interface, constructing even this extension to the existing system was a days work. More complex was actually determining what interfaces could export "system load" in a reliable fashion when a system was actually being severely loaded. This example demonstrates both the capabilities of the underlying STORM system and the benefits accrued from those capabilities. We've found in practice that we can finely control the power demands of applications without extensive system augmentation – this provides a valuable infrastructure to system administrators seeking to reduce operating costs without impacting operations reliability.

Related Work

Virtual machine management has been touched upon by several groups. As we briefly mentioned in our introduction, the focus of STORM is simplicity and ease of infrastucture maintenance. In this section we will compare and contrast STORM to other management systems. VMware currently offers several management products for its hypervisor (ESX Server), the two most relevant to our work on STORM are VirtualCenter (VC), and Distributed Resource Scheduler (DRS) [20].

VirtualCenter is a centralized management tool that allows an administrator to provision, deploy, and manage virtual machines across a cluster of ESX servers. These virtual machines can be custom built or downloaded in an appliance-like fashion. STORM offers the same functionality as VC, however there are key differences in approach. VC assumes that the administrator has a general knowledge of virtualization, while STORM is more designed towards simplicity and assumes no knowledge of virtualization at all. VC and STORM also greatly differ on their approach to Virtual Appliances. Vmware offers appliances that may be manually downloaded from their web site [21], and ran within VC. Unlike STORM, they provide no real distinction between a virtual appliance or machine because of their non-layered approach. An appliance in VC terms contains both the application and operating system, which leads to redundant data.

DRS provides the ability to dynamically allocate virtual machines in a cluster of ESX servers. It will load balance virtual machines based upon utilization. For example, if a virtual machine is allocated a large amount of resources on an ESX server but currently is not using them then DRS will allow other machines to execute on that ESX server. When the load increases, it will adjust accordingly. Using DRS in the scenario we described in our analysis would result in an unresponsive mail server as it cannot increase the amount of resources available to that virtual machine. STORM operates in a similar manner, but offers the ability to address application specific load and scale accordingly by creating more virtual machines with the applications to handle the excess load. In an ideal world, both

STORM and DRS would increase the amount of processors and/or RAM allocated to the virtual machine. While some operating systems [22] in combination with certain hypervisors may offer the ability to dynamically adjust resources, we opted not to restrict STORM to any hypervisor/operating system pair.

Another company taking advantage of this concept is called Enomalism. They have written a web based Xen management system that also has a definition of virtual appliances [23].

There are a number of academic projects focused on managing virtual cluster system. The Collective [8, 7] is a system designed using a metadata-rich specification system; this work is notable for introducing the notion of ''virtual appliances'' and designing a system to manage such appliances. The goal was to manage collections of virtual appliances using the rich CVL (Collective Virtual appliance Langiage). A portion of this project appears to have led to the Moka5 virtual appliance company, which takes a similar approach but focuses on desktop virtualization.

Managing Large Networks (MLN) [10] took a similar approach as the Collective, and used a scripting language (Perl) and extensible metalanguage to configure collections of nodes. MLN was focused on managing *networks* of nodes, and offered a rich configuration infrastructure for that. MLN has been used for projects making use of virtualization for academic infrastructure [24], an application domain we have also targeted. Usher [9] extended this approach to further simplify the management of clusters of related virtual machines.

Most of these system used a common infrastructure (e.g., libvirt) or a similar design. Each used a configuration language – this becomes increasingly important when deploying a *network* of nodes, but is more complex to deploy and manage in smaller appliance-oriented installations.

Although some of the commercial management tools provide integrated power management and scaling options, few of the academic systems have focused on these capabilities. Sandpiper [25] studied the value of different approaches to migrating virtual machines; similar mechanisms would be useful in controlling power, because one goal that we have not implemented would be to ''pack'' virtual machines into as few physical systems as possile. Sandpiper used service level agreement (SLA) specifications to guide their migration strategy; a similar policy specification would be appropriate for power control.

There are fewer projects that have examined *desktop virtualization*; as mentioned, Moka5 is one commercial offering. The Internet suspend/resume Project [26] is one example of a project that has been using virtualization technology to simplify system administration tasks and the way people think about portable personal computing. For example: instead of carrying around a laptop with operating system and applications, the ISR project stores that environment as a virtual machine on the Internet; users carry data with on small device such as a USB drive.

Future Work

There are numerous applications of the STORM system in small to medium business, however it is important to note that there are also several other applications as well. One of the most notable is cluster and datacenter management. In the case of cluster and datacenter management, CPU time could be sold to a customer. The customer would package an appliance designed to execute their application. The appliance would then be uploaded to a channel server, and then it would be install by the datacenter administrators. The STORM system is capable of providing this functionality; however it currently lacks an accounting infrastructure. Creating this functionality is trivial and could be completed in a minimal amount of time. Existing commercial systems such as Amazon's *EC 2* employ similar mechanisms, but the simplicity (and availability) of the Storm infrastructure should allow smaller and regional service providers to offer similar capabilities as similar costs through the automation offered by Storm.

Several additions can be made to the STORM system. Some have already been discussed in this paper, such as an accounting system capable of keeping track of virtual machine CPU usage. This capability could be used to sell time on a datacenter to customers who would find it more cost effective then purchasing and running one of their own. Completedly automated load balancing and service distribution can be added to the STORM system. We demonstrated a version of this capability, but improvements are possible – in particular, accurately estimating load independent of the specific virtual appliance is a difficult task.

In order to accomplish the difficult task of determining need for more resources, or need for more virtual machines assigned to a given task, things such as process load, queue lengths, available I/O operations, and amount of free space in various kernel buffers could be taken account.

For example, a customer running a virtual spam filter suddenly receives a massive amount of incoming spam. This increase causes the spam filter to become overwhelmed. The filter would report this information to the STORM system, which would then respond by either increasing the amount of resources available to the filter or spawning more filters.

Service discovery is another feature that should eventually be added to the STORM system. This would also require client software installed on each virtual machine. It would allow a virtual machine to easily find and contact services provided by other virtual machines. These could be standard services such as

DNS, or services developed specifically for an application.

Since the current system only supports Xen, it would be beneficial to add in support for additional hypervisors; this was the intent of the libvirt project. A virtualization environment can consist of many different hypervisors in order to meet a specific customer's needs. It would also be beneficial to support additional appliance formats. This will allow the customer greater accessibility to a wider range of services, especially ones geared towards other hypervisors.

Conclusions

The resurgence of virtualization has greatly impacted the information technology infrastructure. Companies that lack sufficient knowledge to capitalize on the advantages provided by virtualization are unable to move towards it. The Storm system successfully allows these companies to capitalize on virtualization and reduce or eliminate the need for in house technical support. In general the Storm system allows application developers to provide a single pre-configured virtual appliance to which a customer may deploy virtual machines from. This eliminates the need for each customer to maintain their own operating systems to run the desired application. It does this in a secure and efficient manner by avoiding the common pitfalls that similar solutions suffer from.

We plan to make the STORM system available on SourceForge [27] before the end of 2008.

Author Biographies

Dirk Grunwald recived his Ph.D. from the University of Illinois in 1989 and has been a faculty member at the University of Colorado since that time. He is interested in the design of digital computer systems, including aspects of computer architecture, runtime systems, operating systems, networking and storage. His current research addresses resource and power control in microprocessor systems, power-efficient wireless networking and managing very large storage systems.

Mark Dehus recently graduated with his M.S. in Computer Science from the University of Colorado at Boulder. His focus is virtualization and large-scale systems administration, but is also interested in systems engineering, operating systems, and networking. Some of his current research includes, optimized course capture for web distribution, and applications of virtualization toward computer science education.

Bibliography

[1] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the Art of Virtualization," *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 164-177, New York, NY, USA, 2003.

[2] Khanna, G., K. Beaty, G. Kar, and A. Kochut, "Application Performance Management in Virtualized Server Environments," *10th IEEE/IFIP Network Operations and Management Symposium, 2006 (NOMS 2006)*, pp. 373-381, 2006.

[3] Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. Lam, and M. Rosenblum, *Virtual Appliances for Deploying and Maintaining Software*, 2003.

[4] Varian, Melinda, "VM and the VM community: Past, Present, and Future," *SHARE 89 Sessions*, August, 1997.

[5] Bellard, Fabrice, "QEMU, A Fast and Portable Dynamic Translator," *Proceedings of the USENIX Annual Technical Conference 2005*, p. 41, 2005.

[6] Soltesz, Stephen Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson, "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," *SIGOPS Operating Systems Revew*, Vol. 41, Num. 3, pp. 275-287, 2007.

[7] Sapuntzakis, Constantine and Monica S. Lam, "Virtual Appliances in the Collective: A Road to Hassle-Free Computing," *HOTOS'03: Proceedings of the Ninth conference on Hot Topics in Operating Systems*, p. 10, 2003.

[8] Sapuntzakis, Constantine, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum, "Virtual Appliances for Deploying and Maintaining Software," *LISA '03: Proceedings of the 17th USENIX Conference on System Administration*, pp. 181-194, 2003.

[9] McNett, Marvin, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker, "Usher: An Extensible Framework for Managing Clusters of Virtual Machines," *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November, 2007.

[10] Begnum, Kyrre M., "Managing Large Networks of Virtual Machines," *LISA '06*, pp. 205-214, 2006.

[11] *Turbogears Framework*. http://turbogears.com .

[12] Intel, *Intelligent Platform Management Interface Specifications*, Feb., 2006, http://www.intel.com/design/servers/ipmi/ .

[13] *The Virtualization API*, http://libvirt.org .

[14] VMware, *Open Virtual Machine Format*, http://vmware.com/appliances/learn/ovf.html .

[15] Palmer, Jonathan W., "Web Site Usability, Design, and Performance Metrics," *Information Systems Research*, Vol. 13, Num. 2, pp. 151-167, 2002.

[16] Wagner, David and Bruce Schneier, "Analysis of the SSL 3.0 Protocol," *WOEC'96: Proceedings of the Second USENIX Workshop on Electronic Commerce*, p. 4, 1996.

[17] Paulson, Lawrence C., "Inductive Analysis of the Internet Protocol TLS," *ACM Transactions on Information System Security*, Vol. 2, Num. 3, pp. 332-351, 1999.

[18] *Python 2.5 XML-RPC*, http://docs.python.org/lib/module-xmlrpclib.html .

[19] *Me Too Crypto 0.19 (m2crypto)*, http://chandler-project.org/bin/view/Projects/MeTooCrypto .

[20] *VMware Infrastructure: Resource Management With DRS*, http://www.vmware.com/pdf/vmware_drs_wp.pdf .

[21] VMware, *Vmware Virtual Marketplace*, http://vmwware.com/appliances .

[22] Pinter, S. S., Y. Aridor, S. Shultz, and S. Guenender, "Improving Machine Virtualization with 'Hotplug Memory'," *17th International Symposium on Computer Architecture and High Performance Computing*, pp. 168-175, Oct., 2005.

[23] Enomolism, Inc., *Enomolism Elastic Computing Platform*, http://enomolism.com .

[24] Gaspar, Alessio, Sarah Langevin, and William D. Armitage, "Virtualization Technologies in the Undergraduate IT Curriculum, *IT Professional*, Vol. 9, Num. 4, pp. 10-17, 2007.

[25] Wood, Timothy, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif, *Black-Box and Gray-Box Strategies for Virtual Machine Migration*.

[26] Satyanarayanan, Mahadev, Benjamin Gilbert, Matt Toups, Niraj Tolia, Ajay Surie, David R. O'Hallaron, Adam Wolbach, Jan Harkes, Adrian Perrig, David J. Farber, Michael A. Kozuch, Casey J. Helfrich, Partho Nath, and H. Andres Lagar-Cavilla, "Pervasive Personal Computing in an Internet Suspend/Resume System," *IEEE Internet Computing*, Vol. 11, Num. 2, pp. 16-25, 2007.

[27] *Sourceforge: Open Source Software Development Web Site*, http://sourceforge.net .

IZO: Applications of Large-Window Compression to Virtual Machine Management

Mark A. Smith, Jan Pieper, Daniel Gruhl, and Lucas Villa Real – IBM Almaden Research Center

ABSTRACT

The increased use of virtual machines in the enterprise environment presents an interesting new set of challenges for the administrators of today's information systems. In addition to the management of the sheer volume of easily-created new data on physical machines, VMs themselves contain data that is important to the user of the virtual machine. Efficient storage, transmission, and backup of VM images has become a growing concern. We present IZO, a novel large-window compression tool inspired by data deduplication algorithms, which provides significantly faster and better compression than existing large-window compression tools. We apply this tool to a number of VM management domains, including deep-freeze, backup, and transmission, to more efficiently store, administer, and move virtual machines.

Introduction

Virtual machines are becoming an increasingly important tool for reducing costs in enterprise computing. The ability to generate a machine for a task, and know it has a clean install of a given OS can be an invaluable time saver. It provides a degree of separation that simplifies support (a major cost) as people all get their "own" machines [26].

A side effect of this, however, is that there can be many machines created in a very short time frame, each requiring several gigabytes of storage. While storage costs are falling, terabytes are not free.

We believe that large-window compression tools provide much needed relief in this domain, because they often perform extremely well on large data sets. Large-window compression has not received much attention in the past because archiving tasks have traditionally involved only hundreds of megabytes. Moreover, traditional tools such as gzip perform consistently well for small and large data sets. However, falling storage costs and larger storage applications (e.g., virtual machines) have driven archiving to hundreds of gigabytes or even hundreds of terabytes. Large-window approaches provide significant additional compression for data sets of these sizes, and complement small-window compression.

One of the most exciting results of our experiments is that large-window compression does not impact the compression factor achieved by a small-window compressor. The combined compression ratio is close to the product of the individual compression ratios. In some cases we even noticed small improvements in the compression ratio of gzip when applied to data that had been processed by our large-window compression algorithm. Moreover, the total time required for large-window and small-window compression combined is often smaller than that of the small-window compression alone.

We find this property analogous to the way a suitcase is packed. It is possible to simply stuff all of one's clothes into a suitcase, sit on the lid, and zip it up. However, if the clothes are folded first, and then compressed, they will fit in a smaller space in less time. Large-window compression can be thought of as the folding step, while small-window gzip style compression is the sitting and zipping. Using the term compression for both techniques can become confusing, so in such cases, we refer to large-window compression as data folding.

In this paper, we present three applications of large-window compression for the efficient management of virtual machines. We also introduce IZO, a novel large-window compression tool using data deduplication, which accomplishes significantly better data folding than existing large-window compression tools.

The first application is the use of large-window compression in cases where long-term storage of "retired" machines is desired. This is a common case when a group of machines have been generated for tasks that are no longer active, but which may become active again. Examples include yearly activities, or machines for people who are only present periodically (e.g., summer students). In this scenario a number of similar machines are to be placed in "deep freeze."

The second task is that of distributing machine images. Core images of machines are often used for the "base machine" with users having their own mounted filesystems. The core image is usually maintained by experienced IT staff, e.g., ensuring that the latest security patches and drivers are installed properly. In a global organization these new images need to be distributed to remote sites. For example, in the financial sector it may be necessary to distribute updated machine images to all bank branches. Unfortunately, these images can be quite large. A problem that can be compounded by low or moderate Internet connectivity for remote sites. We observe that each image tends to be very similar to the previous one. We therefore investigate whether a smaller patch file can be created by calculating the binary delta between similar virtual machines. We use the series of RedHat Enterprise Linux installs from version 4 to create a set of virtual machines from the initial release to updates 1-6 and calculate binary deltas between these machines.

The last task is in creating regular backups of these systems. Traditional backup tools need to be installed on the actual machine and require the machine to be running during the backup. They are often specialized on a specific operating system and filesystem. One benefit of these specialized solutions is that they provide incremental backups: the ability to identify which files were modified since the last backup, in order to save storage space on the backup device. This can be difficult if the virtual machines are running a variety of (potentially arcane) operating systems or filesystems. We instead look at creating backups of the whole virtual machine image, and using large-window deduplication to identify the changes between different revisions of the backup, to achieve the footprint of incremental backups without specialized knowledge of the inner workings of these virtual machines.

In all these cases there is considerable redundancy or duplication of data – for example, files that appear on multiple filesystems. These duplicates may occur gigabytes apart in the archive, and thus traditional streaming short-window compression schemes such as gzip are unlikely to "see" both duplicates within their window, and therefore cannot compress them.

We are working on a novel compression tool called IZO (Information Zipping Optimizer), which was initially targeted at compressing ISO CD images. IZO uses data deduplication algorithms to identify and remove global redundancies. Because large-window compression tools are expected to be applied to hundreds of gigabytes of data, we desire reasonable memory requirements and fast data processing. Moreover, a scriptable interface is important to allow automation of the different compression scenarios. For IZO, the command line interface has been modeled after tar, providing intuitive ease-of-use to users that are familiar with this tool. IZO also provides the same streaming semantics as tar – it requires a set of files or directories as input, and can stream the compressed output to stdout, or to a specified archive. IZO only removes large-window redundancies, therefore the output should be streamed into a short-window compressor of the user's choice.

In the remainder of this paper we will examine some background on the compression and data deduplication approaches used (Section Background), outline the scenarios we are examining (Scenarios), provide a brief tutorial on large-window compression and outline our implementation (Tutorial and Implementation), discuss the experiments and their results (Experiments) before concluding with some thoughts on how these tools can be extended even further (Conclusions and Future Work).

Background

Virtual machines are an emerging tool for allowing multiple "guest machines" to be run on a single piece of hardware. Each guest thinks it has a whole machine to itself and the host machine juggles resources to support this. A full discussion of the challenges and opportunities of virtual machines are beyond the scope of this paper, but the reader is encouraged to consult [19, 28, 21] for a good overview. A good introduction to the open source virtual machine monitor Xen can be found in [5, 6] and the commercial product VMWare in [27, 22].

We are working on a novel large-window compression tool called IZO. IZO draws on concepts from existing tools and technologies including tar [7], gzip [9], rzip [24, 25] and data deduplication [29, 14]. It is invoked on the command line using parameters identical to the well-known tar utility to produce an archive of multiple files or directories.

Large Window Compression

Traditional LZW-style compression techniques are notoriously poor at handling long-range global redundancy. This is because they only look for redundancy within a very small input window (32KB for gzip, and 900KB for bzip2 [20]). To address this shortcoming, rzip [24] was developed to effectively increase this window to 900 MB. Rzip generates a 4-byte hash signature for every offset in the input file and stores selected hashes as {hash, offset} tuples in a large hash table. Data at offsets that produce the same hash signature may be identical. The data is re-read and a longest-matching byte sequence is calculated. If a matching byte sequence is found, a redirecting (offset, length) record is written to the rzip file instead of the duplicate data. This method achieves extremely high compression ratios, but suffers from two main drawbacks as described by rzip's author Andrew Tridgell in [24]. First, rzip has large memory requirements for storing a hash table entry for every offset in the input file. Second, it cannot operate on input streams because it requires random access to the input data to re-read possible matching byte sequences.

Rzip also compresses the deduplicated archive using bzip2 at the highest compression level. Unfortunately, this step may require significant resources and time. Lrzip [10] provides an extended version of rzip, allowing the user to chose between different short-range compression schemes or to execute rzip's long-range compression only. In the experimental evaluation, we compare IZO to lrzip, because lrzip allows us to assess the benefit of rzip's long-range compression in isolation.

Even though IZO achieves large-window compression as well, its roots are quite different. It was inspired by our work on the StorageNet deduplication filesystem [18, 23]. Data deduplication filesystems are a recent trend in enterprise class storage systems, such as NetApp ASIS, DataDomain, Sepaton DeltaStor and Diligent HyperFactor. IZO was created on the insight that the deduplication algorithms used in these filesystems could provide data compression as a command line tool similar to tar or gzip as well.

CZIP [13] takes a similar approach, storing compressed data in a single archive file. But it focuses on efficient transmission of data over a network by maintaining a "Content-Based Named" CBN dictionary of chunks, which has to be stored in the archive as well. CZIP enables Servers to prevent cache-pollution by using this dictionary to only load a unique instance of a particular chunk, while clients can decrease network transmission by re-using already-sent duplicate data chunks. IZO, in contrast, is designed as an offline compression tool. Creating the smallest possible archive from its input is its primary goal and thus we perform optimizations such as discarding the CBN dictionary, etc. to reduce the total compressed size. The use of a CBN dictionary also precludes CZIP from being able to stream both into its compressor and decompressor. To stream into the compressor, the CBN dictionary must be written to the end of the archive; while to stream into the decompressor, the CBN dictionary must be at beginning of the archive.

Data Deduplication

Data deduplication can eliminate global redundancies across multi-terabyte-scale corpora [11]. Early deduplication filesystems include Deep Store [30] and Venti [16], which are also called content-addressable storage systems.

These systems identify duplicate data when it is written to the filesystem. They then reference the existing data instead of storing another copy. Duplicates can be detected on a whole-file level, but current solutions usually detect sub-file redundancies. This is done by breaking files into chunks. A chunk signature is created using cryptographically strong hashing of the chunk content to avoid collisions. Chunks with the same hash value are considered duplicate, and only a single instance is stored. Files are comprised of a list of unique data chunks and are reconstructed during read. The list of unique hashes needs to be kept in memory, to provide fast lookup for newly incoming data. Therefore, chunk sizes in these systems are generally large, typically between 4KB and 32KB.

Different methods exist to split the data into chunks. The chunking method and chunk size strongly influence the speed of the system and the overall data compression. A detailed discussion of chunking approaches can be found in the data chunking section below.

Commercially available deduplication products use a wide variety of chunking approaches. NetApp ASIS uses a modified version of WAFL to deduplicate at a 4K fixed-block granularity [8, 12]. Fixed-block chunking is very fast, but provides comparatively low folding factors. Data Domain's Global Compression uses content-agnostic chunking to produce chunks of 8KB average size [2]. Content-agnostic chunking is based on a rolling hash function, such as Rabin Fingerprinting [17] and provides better folding factors, but at the expense of speed. Sepaton's DeltaStor is content-aware, with built-in knowledge of common data formats for optimal chunking [3]. This is the slowest method because data must be interpreted, but can provide the highest folding factors. HyperFactor, by Diligent, departs from the typical chunking methodology, and relies instead on using data fingerprints to identify similar data objects, and stores the differences between them [1]. Computing the differences requires a read of the most-similar identified data object, but because chunk hashes are neither computed nor compared, this technique does not suffer from the possibility of hash collisions.

Storing virtual machines on a deduplication filesystem would provide similar benefits as the ones we hope to accomplish in our scenarios. Moreover, live filesystems allow virtual machines to be deduplicated even when they are running. However, these benefits are lost when a set of machines is moved to a non-deduplication storage device, such as tape or a different disk array. We are interested in evaluating the usefulness of a simple deduplication compression tool that can be used on a standard desktop and doesn't require the acquisition of specialized storage hardware. IZO bridges this gap by providing a deduplication archive, which can be stored on any storage device without loss of compression.

Binary Diff

It is curious to note that the algorithms employed by rzip and data deduplication can also be used to calculate the differences between two files. The difference is defined by all data in the second file that is found to be non-duplicate with regards to the first one. Binary deltas between two versions of an executable are useful to create patch files that are significantly smaller than the new executable. Our VM distribution scenario in Section "Scenarios, VM Distribution" evaluates the ability of IZO to create binary diffs. The equivalent tool from the rzip family is xdelta [24, 4].

Scenarios

In all the following scenarios we are trying to put ourselves in the "mindset" of a systems administrator for a virtual machine farm. We want approaches that reduce the cost/effort of maintaining such systems, are simple enough to be useful within seconds of reading the man page, are scriptable, and hopefully work well with existing infrastructure. To this end we use gzip for small-window compression, and IZO with its "tar"-like syntax for large-window. As such it can be used as simply as

```
izo -cv /mnt/vmimages | \
        gzip > freeze.igz
```

Deep Freeze

A not uncommon scenario with virtual machine management is to have a number of machines that are no longer being used. However, the data on them is too valuable to just throw away. We've begun to archive working demo systems at the end of project life cycles as virtual machines. These systems provide self-contained "time capsules" of code, libraries and configuration that can be "revived" at a later time.

The desire is to take a set of similar machines and deep freeze them – that is compress the set as much as possible, and then place them on low-cost storage (potentially near or off line). If they are ever needed again it is understood that it may take a day to restore them, and the move to deep freeze can be done in batch during "off time." If there are a number of machines that need to be frozen they may be done in batches, or they may trickle out (e.g., as summer students leave). Thus we would also like to be able to "append" new machines to the end of a freeze.

As a proxy for this, we use images of machines from a set of computers generously provided by a local after school program. The 11 machines all started life with similar installs of Windows XP, but have "evolved" over the course of a year of student use. We look at this set for examples of how a group of similar but not identical machines can be deep frozen together.

Deep freeze is most effective if the machines in each freeze set are reasonably similar. They should, for example, have the same underlying operating system and preferably be used for similar tasks. Fortunately this is usually the case in enterprises.

VM Distribution

One common use of virtual machines is to provide many employees with their "own" machines, on which their home directory is mounted but on which they have limited permissions to modify the rest of base machine. This simplifies the support task, as a central IT department can create and test the image which is then distributed. For global organizations this last step is non-trivial. If the images are generated in South East Asia, the testing is done in North America, and the images are deployed to company branches across Europe, sending multi-gigabyte images around can become quite cumbersome – especially when network connectivity to some sites is slower than may be hoped.

We find, however, that the differences between these base images can be quite small. Using the original image as a "template," the updates then become "deltas." We use RedHat Enterprise Linux 4 WS as our test image for this scenario. As a framework, we used Xen on a Fedora Core 7 machine, but since we were using hardware virtualization from the processor and raw files for disk images the same results can be achieved using other VM solutions. We installed each operating system on a 10 GB partition answering "yes" to all questions, starting with the base RHEL4 and then doing the same for the updates u1 through u6. We then create the binary delta between the initial RHEL4 image and the RHEL4U1 image; the RHEL4U1 image and the RHEL4U2 image; and so on.

The goal then is to create incremental patches that can be as small as possible, reducing the "cost" of an update and thus encouraging shorter and less cumbersome update cycles – critical in the case of security or machine stability issue patches.

Backup

Virtual machines give tremendous freedom to IT professionals. With the emergence of full hardware virtual machine support in both Intel and AMD chip sets, it is possible for users to run whatever operating system and programs best support their tasks. This freedom comes at a price. Inevitably someone will be running OS/2 Warp with the HPFS filesystem.

Despite the wide variety of operating systems, versions, usage scenarios, etc. there is still an expectation that system administrators will provide basic support (such as system backup and recovery). Support for modern backup clients in OS/2 Warp is unfortunately limited, and even mounting the HPFS filesystem can prove problematic. We thus need a way to backup such machines and not rely on the users to maintain copies of important files.

This is facilitated by the proliferation of snapshot functionality – either at the storage level in NAS/SAN products, or in software such as the LVM system. These facilities allow a snapshot to be taken of a running system which can then be saved as a backup.

This snapshot is essentially a disk image file, often many gigabytes large. Keeping a snapshot from every week is an expensive proposition. What is desired is the ability to do a binary "diff" of each image and store only the changes. This is thus similar to the former task, although the data evolution is quite different. Additionally, real savings can be obtained by performing snapshots of a number of machines (as per the "deep freeze" scenario).

We will look at backing up a single user's machine every week for a number of weeks. The backups are full filesystem images of a machine running Windows XP, including all data, application, and system files. The machine is one of the authors' actual workstations, used every day during the work week for email, office applications, and software development, and thus is typical of the kind of machine one might find in a corporate environment.

We repeat this experiment with a virtual machine running OS/2 Warp 4. This machine is not actively being used, but different snapshots are taken after modifying the filesystem by adding and removing files.

Tutorial and Implementation

Large-window compression/deduplication is a fairly new technology to most IT professionals, so we will review the basic idea behind it, and the choices we made for our IZO implementation. For those already familiar with large-window compression (and/or not interested in the implementation details) skipping to the next Section might be advisable.

IZO operates like an archiver with built-in data deduplication facilities. To perform data deduplication and create archives, IZO implements three core components: a chunking algorithm, a chunk hash table, and an archive format generator.

Data Chunking

As described earlier, the core technology choice in a deduplication implementation is the method by which the data is chunked. The major chunk generation schemes are as follows:

Fixed-Size Chunking: Fixed-size chunking breaks data into chunks of a specific size. It is very simple, fast, and the resulting chunk size can be selected to optimally align with the physical block size of the underlying storage device to maximize storage utilization. However, a major drawback of fixed-sized chunking is that shifting the data by some offset may result in completely different chunks. This issue is illustrated in Figure 1.

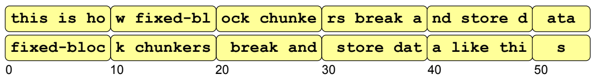


Figure 1: Fixed-size chunking.

Content-Aware Chunking: This method, illustrated in Figure 2, generates chunks by parsing the input files and understanding their formats. Chunk boundaries can be custom generated for the highest probability of redundancy. In this example, boundaries are created between words. This method does not suffer from byte-shifting issues, and can produce larger chunks, but is useful only for known data types and can be very expensive because the data must be interpreted.

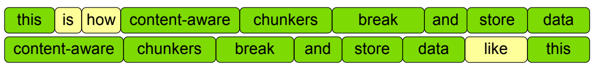


Figure 2: Content-aware chunking.

Content-Agnostic Chunking: Content-agnostic chunkers use the content to determine chunk boundaries, but do not understand the content format. Instead, they consider the features of the byte sequence to deterministically identify boundaries.

This is usually done by generating a rolling hash over a window of the input stream, e.g., using Rabin fingerprinting [17]. Each hash is masked and the resulting value is placed through a modulus operation. If the result of the modulus is zero, then a chunk boundary is created. The modulus value determines the frequency of chunk generation. A modulus of 16, for example, would produce an average chunk size of 16 bytes. Chunks produced this way have a high probability of being duplicates with other chunks produced using the same method. This is because all such chunks are guaranteed to end with a byte sequence that generated modulus-value bits of zeros in the rolling hash. This method eliminates the byte-shifting issue that fixed-size chunking suffers from, but is more expensive because of the hashing.

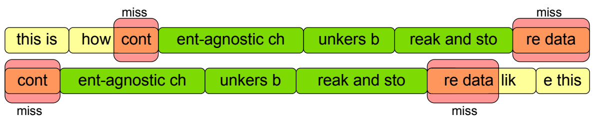


Figure 3: Content-agnostic chunking.

This method is also subject to "external fragmentation" wherein matching chunks may have a small number of bytes either before or after the chunk boundaries that are also identical, but not necessarily detected. This can be seen in Figure 3 in which the missed duplicate byte fragments are highlighted. Finally, because the chunks are variable size, this can lead to wasted storage on the physical level, for example, a 3KB chunk requires a whole block in a 4KB filesystem, wasting 1KB.

For IZO, fixed-block chunking may not provide enough deduplication, and content-aware chunking is too expensive and does not generalize well. Aligning chunk sizes with the physical disk is unnecessary, because IZO produces just a single output stream, which will be optimally stored by the underlying filesystem. Our prototype therefore uses content-agnostic chunking in order to provide maximum deduplication. Although still subject to external fragmentation, we implement an optimization (described in detail in the data format section) in which contiguous chunks can be merged into superchunks in the reconstruction metadata.

Chunk Hash Lookup

Deduplication filesystems require the fast lookup of chunk hashes to determine whether a data chunk exists in the system or not. This is one reason why these systems generally use large chunk sizes: the storage overhead of this metadata increases rapidly for smaller chunk sizes and lookup performance plummets as the data cannot be cached in memory anymore. On the other hand, smaller chunks allow the detection of more duplicates, increasing the deduplication ratio.

For the implementation of IZO it is possible to discard the in-memory hash data once the compression is completed. The main drawback of not storing this metadata in the archive is that adding additional files to the archive is not possible or computationally expensive. Without this information, new chunks cannot be directly compared to the existing chunks. However, the original hash table can be rebuilt by re-chunking and re-hashing the existing archive. Another option is to store this metadata in a separate file so that additions can be made quickly, but consumers of the archive are not burdened by the size overhead.

Not having to store the hash data reduces the final archive size. For example, a 1 GB input file deduplicated at a 512-byte average chunk size and achieving a folding factor of 2 (deduplicated to 500 MB) would produce one million 24-byte {hash, offset} tuples, accounting for 24 MB or 5% of the archive size. Not storing this metadata also allows us to use much smaller chunk sizes because we won't have to worry about "polluting" the output file with inflated hash data. However, the reconstruction metadata (the {hash, offset} tuple list) still grows with decreased chunk size. For very small chunk sizes it is worse than we would wish, due to increased segmentation of the data. Figure 4 shows the amount of metadata from our deep freeze experiment for a single VM image (37.3 GB) as input. The upper line shows the expected growth of a factor of two. The reconstruction metadata (lower line) roughly follows this line until a chunk size of around 512 bytes. At 256 byte the metadata grows even faster, because increased segmentation requires more reconstruction information.

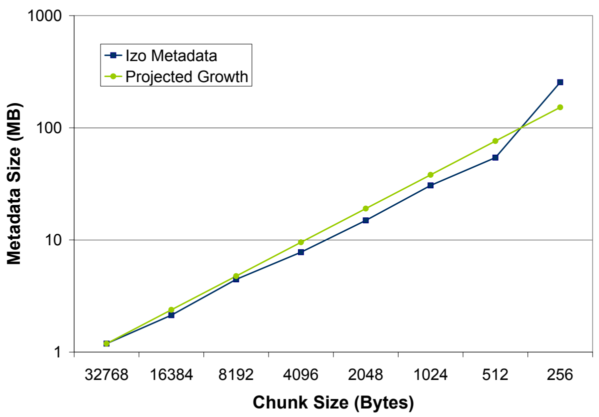


Figure 4: Effects of chunk size on reconstruction metadata.

Chunk Hash Collisions

By relying on the chunk hash to guarantee chunk matches, IZO is able to operate on sequential input streams. This is because IZO does not need to seek back in the stream to verify that chunks are identical. Unlike rzip, which uses a 4-byte hash as a hint that the indicated data may be identical, IZO uses a 16-byte MD5 hash to probabilistically guarantee that the indicated chunks are the same. A detailed account of the probability of a hash collision in a data deduplication system is provided for EMC's Centera [15].

The problem of a hash collision occurring between two chunks in a deduplication system simplifies to the birthday paradox (see Appendix). In the case of IZO, the namespace n is the number of unique 128-bit hashes (2^{128}), while the number of things to be named c is the number of chunks in the archive. With an archive of 1 TB and an average chunk size of 1 KB, c is 2^{30}. For large c's, the collision formula simplifies: $c^2/(2n - c)$. For an archive of 1 TB using an average chunk size of 1 KB, the probability of collision is 1 in 2^{68}. For a 1 PB archive, the probability is 1 in 2^{48}. For comparison, the non-recoverable Bit Error Rate (BER) of an ATA hard drive is 1 in 10^{15} (2^{50}) [15].

The IZO Data Format

The design of the IZO data format was driven by the desire to pack data and metadata as tightly as possible into the resulting archive. Instead of storing a list of chunk identifiers for each file, we use offsets and lengths within the archive. In particular, if a sequence of chunks from the original file is new to the system, then it is being stored as the same sequence into the output file, requiring only one {offset, length} tuple. Because we keep track of all individual segments during compression, it is still possible to match subsequences within this data.

The hash metadata is not being persistently stored. The only metadata that is required is the path and filename for each file in the archive, along with file-specific information such as ctime, mtime, etc., and a list of {offset, length} tuples to reconstruct the constituent files.

It should be noted that we create an IZO archive in segments. A segment remains in memory until it is completely prepared, and is then streamed out. This allows us to perform random updates within the segment (such as to the segment length) without the need to seek on disk. A typical segment size is 8 MB. File data can span segments, but each new file to the archive begins a new segment. Reconstruction references in a given segment are always relative to the beginning of the segment in which they sit. These references always refer either to chunks in the same segment (positive values), or to chunks in a previous segment (negative values), but never to chunks in a future segment. This property was designed into our data format to provide for appendability of archives. Just as with tar, one can concatenate two IZO archives to form another valid IZO archive.

We demonstrate the IZO file format with an example of processing two input files shown in Figure 5. The processing steps are illustrated in Figure 6. The first four bytes of the IZO archive store a 32-bit segment length. The segment length is not known until the entire segment is populated. Immediately following the segment length is the reconstruction metadata. The reconstruction metadata contains information such as the file name and a list of {offset, length} tuples to reconstruct the file. Following the reconstruction metadata are the actual data chunks from the first input file (see Figure 6(a)). As unique chunks are encountered, their hashes and offsets are stored in the in-memory hash table and the actual chunk data is written to the current segment in the archive.

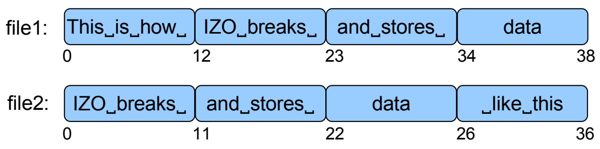


Figure 5: Example input files for IZO.

Once the first file has been processed, the lengths of both the reconstruction metadata and data are known, and we can update the segment length field and stream out our segment (see Figure (b)). In this example, the reconstruction metadata is placed at offset 4, and contains only one {offset, length} tuple, because the first file did not provide any duplicate chunks and was therefore written as-is. The file1 reconstruction metadata specifies that to recreate file1, read 38 bytes from offset 22 relative to the beginning of the segment (0+22=22).

Now, IZO processes the second file. The first chunk matches the chunk stored at offset 34 with length 11. The next chunk matches at offset 45 with length 11. Because these two chunks are in sequence, IZO uses the superblock optimization and only modifies the length of the matching data section from 11 to 22. The third chunk is also a contiguous duplicate of four bytes at offset 56, resulting in another update to the length to 26. The last chunk is new to the system and is added to the archive, and its hash is added to the in-memory hash table. Finally, the metadata length for the second file is known and the segment is finalized and streamed out. Looking at the reconstruction metadata for file2, it specifies reconstruction of the file by reading 26 bytes from an offset 26-bytes before the current segment (60-26=34) and then 10 bytes from an offset 34 bytes after the beginning of the current segment (60+34=94) (see Figure 6(c)).

Experiments

We performed our experiments on a 2-way 2.0 GHz 64-bit Intel Pentium Xeon server with 2 GBs of DDR RAM with four 500 GB 7200 RPM UDMA/133 SATA disks of storage. This appliance hosts a SUSE SLES64 version 10.1 operating system which is built upon Glibc 2.4-31.30 and GCC 4.1.2. The operating system and all executable were installed on the first disk, while the remaining three disks were used for data storage.

Tuning IZO

We first examined the effect of the block size that IZO would use for the deep freeze experiment. We tried average variable-sized blocks ranging from 32 KB in size down to 256 bytes. We ran each block size against first a single 40 GB virtual machine image, then added the next, and so on up to all 11 images.

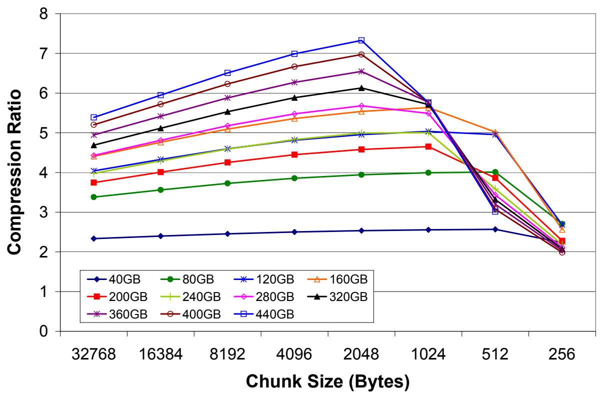
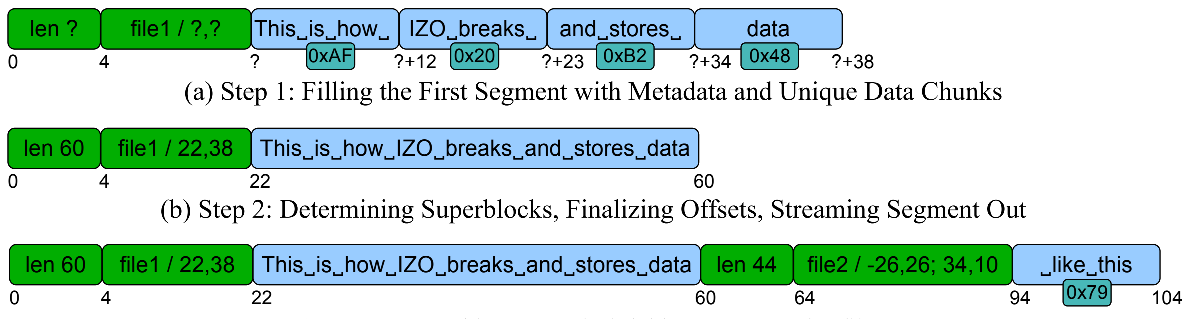


Figure 7: Effects of chunk size on combined compression ratio.

Figure 7 shows the overall fold+compress ratios for these image sets at each block size. The best overall compression is garnered from 2 KB block sizes. In our experience with this tool, generally we find that with large datasets, in the hundreds of GBs for



(a) Step 1: Filling the First Segment with Metadata and Unique Data Chunks

(b) Step 2: Determining Superblocks, Finalizing Offsets, Streaming Segment Out

(c) Step 3: Working on and Finishing Segment for file2

Figure 6: Output creation.

example, block sizes in the range of 2 KB to 8 KB yield a good folding factor. When dataset size is smaller than 10 GB or so, the block size can be reduced, with 128 bytes as the lower limit.

Smaller blocks can have a negative impact on the folding factor for two reasons. First, smaller blocks can greatly increase the amount of metadata required to store references to previously-seen blocks. Second, smaller blocks more quickly fill the in-memory hash table we use to store references to all of the blocks in the system. With a large dataset, the hash table becomes full, and we must start evicting block references, becoming effectively blind to a number of previously seen blocks. For these reasons, in our subsequent experiments we use a 2KB block size for IZO.

Deep Freeze

The deep freeze experiment uses a number of similar but not identical machines and tries to reduce the amount of storage the set of images will require. For this experiment we look at large-window deduplication (IZO), large-window compression (lrzip) and small-window compression (gzip).

The dataset for this experiment was eleven 40 GB Windows XP drive images of machines that started with a similar install but then diverged for a year in a community after-school program.

Table 1 shows the archive size and time required for gzip, IZO and lrzip. Using the large-window deduplication tuned to operate at a 2KB block size on a typical XP system results in a folding factor of 1.92 for just a single machine. This compares to the gzip-only compression ratio of 1.40. These two techniques are more or less orthogonal and can therefore be combined. Applying gzip to the IZO-archive provides an overall compression ratio of 2.54.

Timewise it is clear that gzip is fairly intensive – running it on the smaller IZO file instead of on the original data results in time savings of 22%. For a single image we see that deep freezing (fold+compress) takes 43 minutes, and will result in an overall storage savings of 61%. This compares to gzip alone on the original data, which takes 55 minutes and yields a savings of only 28%. lrzip does not find much duplication, and hands a large output to gzip, resulting in a 97 minute effort to save a combined 36%. So, not only does the fold+compress technique yield the best overall compression ratio, it is also much faster than the next fastest technique.

Where deep freezing really shines is when we examine freezing more than one image at the same time. In these cases IZO can exploit cross-image similarities to garner much higher folding factors. For five images, IZO+gzip affects a 78% storage reduction in 144 minutes. Applying lrzip+gzip yields a data reduction of 36% and takes 527 minutes, longer than an 8-hour working day.

In the case of 11 images IZO achieves a folding factor of 5.47, and gzip is able to push it 7.32, a storage savings of 86%. This takes only 262 minutes, more than 2 times faster than running lrzip+gzip on only 5 images. We do not provide measurements for lrzip on eleven images because of the long run-time this would require.

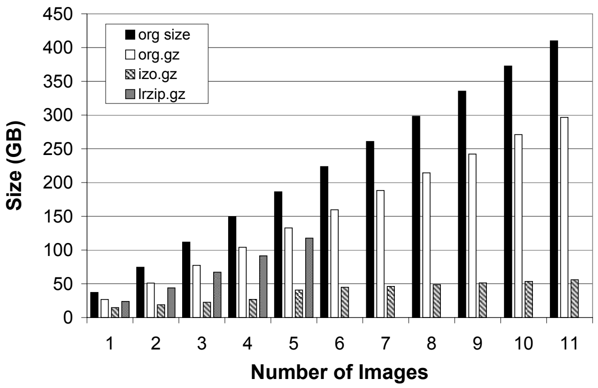


Figure 8: Compression of multiple Windows XP images.

Figure 8 illustrates these finding across all of the machine images. While the original size (in black) and gzip size (in white) continue to increase rapidly with the number of images, IZO+gzip (in stripes) grows much more slowly. lrzip+gzip does slightly better than gzip alone, but does not approach the IZO+gzip ratios.

The reason that IZO is able to achieve such high folding factors on this data, and lrzip is not, has to do with the effective window size of the two large-window compression tools. In order to take advantage of any inter-image duplication at all, a tool must have a window at least as large as a single image, in this case, roughly 40 GB. To eliminate duplication across all 11 images, this window must encompass every image, over 400 GB in our case. lrzip has an effective window of around 900 MB, and is therefore unable to "see" and eliminate the inter-image duplication. IZO, with a 2 KB average block size, has an effective window in the TB range, and is therefore able to deduplicate across the entire set of 11 images.

Number	Size (GB)						Time (minutes)				
Images	orig	gzip	izo	izo.gz	lrzip	lrzip.gz	gzip	izo	izo.gz	lrzip	lrzip.gz
1	37.3	26.7	19.4	14.7	30.6	23.9	55	14	43	49	97
5	186.4	132.8	51.7	40.7	151.4	117.7	282	67	144	269	527
11	410.0	296.6	74.9	56.0	–	–	556	153	262	–	–

Table 1: Deep freeze scenario compression numbers using 2KB variable chunks.

VM Distribution

This scenario simulates the case when a company needs to update a set of standard images, which may be distributed globally to different remote offices or branches. Traditionally, images are either simply gzip compressed, or a delta is computed between the two images, and the delta is compressed. In the case of creating deltas, the original image is assumed to be at the destination already. We investigate the use of IZO to produce a folded delta that can be applied to the original image to produce an updated image.

For this scenario we used seven "clean installs" of RedHat Enterprise Linux Workstation version release 4 to simulate incremental upgrades of a base system that might be used in a virtual machine environment. We have one install for each update (u1-u6) plus the original release. Each install is done on a 10 GB partition, the majority (roughly 7.5 GB) of which is sparse.

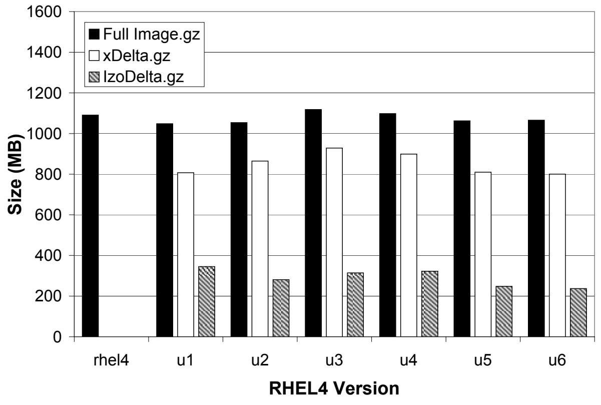


Figure 9: Binary deltas between incremental VM versions.

Figure 9 shows the smallest deltas that can be achieved between successive updates to the previous image. u1 represents the delta to the base, u2 represents the delta to u1, and so on. As illustrated, the overall image size after gzip compression is just over a gigabyte. Already this is a significant savings due primarily to the sparseness of the 10 GB image. When xdelta is applied to the images first, and then gzip is applied, the data size is reduced to an average size of 851 MB, or an additional 21% on average. IZO, however, provides an additional 73% storage savings over gzip alone. This reduces the amount of data to an average of 290 MB that is needed to transform one 10 GB image into its subsequent version – just 3% of the entire image size, and roughly three times smaller than xdelta+gzip .

Backup

In this experiment, we look at the application of large-window compression techniques to weekly backups of a single user's machine. The goal is to provide a traditional incremental backup of the machine without requiring any end user action, nor the ability to install clients on the end user machine.

The backups in question are full filesystem backups of a machine running Windows XP, including all data, application, and system files. The machine is one of the authors' actual workstation, used every day during the work week for email, office applications, and software development.

The backup schedule used by this machine was very simple. It completes the first backup of roughly 23 GB on Sept 8, and then performs a full weekly backup for eight weeks. This scenario is analogous to the scenario described in the introduction, wherein a user may be running an operating system in a VM without palatable means of backup from within the system. By running large-window compression over the entire system periodically, the system VM administrator is able to effectively perform multiple full system backups in an operating-system agnostic way, while achieving smart incremental backup space efficiency.

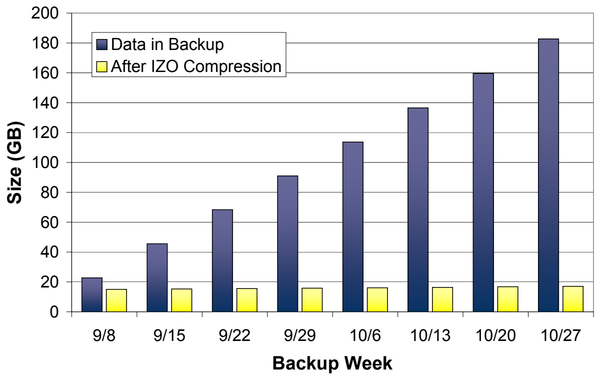


Figure 10: Compression of weekly system backups.

Based on our previous experiments, we chose to use a content-agnostic chunker based on Rabin fingerprinting to produce blocks of variable size of around 2 KB for this test. The results we observed from these experiments were very encouraging for this method of backup (see Figure 10). The first observation was that even for the very first backup, the size of the data was reduced from 23 GB to 15 GB (note, for these experiments we have not gziped the resulting IZO files). Adding successive backups (using the append functionality of IZO) consistently showed that very little additional data was added to the system. In fact, during the two months of use of this machine, the user added just less than 1 GB of unique data. Including all data and metadata, IZO is able to store all 8 full filesystem backups in 17 GB, which is smaller than the original 23 GB image itself.

It is interesting to note where the growth in the IZO archive actually occurs (see Figure 11). The IZO archive grows from 15 GB to store a single image to 17 GB to store all 8 images. However, the growth of the actual data stored in the archive grows even more slowly. The predominant growth of the data can be attributed to the reconstruction metadata used to point back to the data in previous backups. We are currently investigating ways to compress the metadata itself, by looking at similarities in the metadata between different backups.

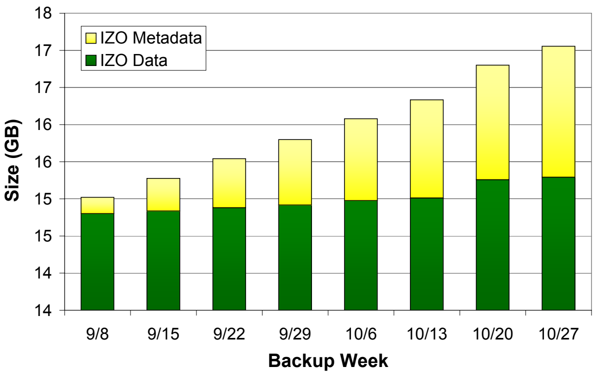


Figure 11: Increase in the amount of meta data stored in IZO for weekly system backups.

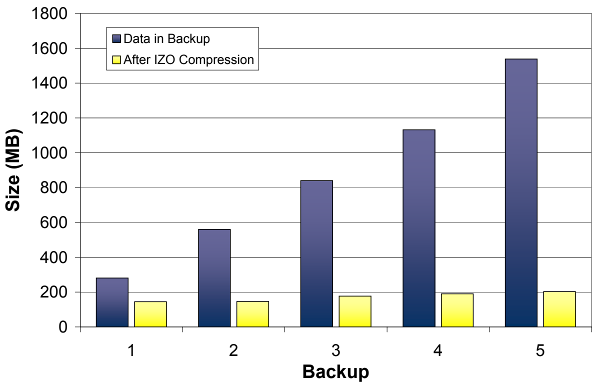


Figure 12: Compression of a collection of OS/2 VM images.

These numbers are for a single machine. The advantages to performing multiple machine backups into the same IZO archive should be comparable to the advantages seen in the deep freeze experiments.

Next, we studied as a proof of concept the use of deduplication to enable backup of a VM running a somewhat uncommon OS. The idea is to verify that such machines can be backed up, even if there is no client or filesystem support for them. We created a virtual machine using Microsoft VirtualPC and OS/2 Warp 4. We then ran the machine and performed simple filesystem modifications in OS/2 by adding data to the disk or removing files from disk. Once in a while, we stopped the machine and created a "backup" by copying the VM image. It is clear that this approach does not fully capture the change that would occur during real-world use of an OS/2 installation, but it provides some insight in whether IZO can achieve incremental backup space efficiency for OS/2 as well.

Figure 12 shows the results of compressing up to five backups. The initial VM image is only 281 MB, and IZO is able to compress the image to 145 MB already. Adding additional backups slowly increases the overall archive size to 203 MB. During the experiment, we added a total of 93 MB of new data and removed 32 MB of data from the OS/2 filesystem. Our backup #5 encodes all five backups and only requires 58 MB of additional space over the base image, clearly providing incremental backup functionality.

Conclusions and Future Work

Large-window compression is a fairly new technology that has many applications to the domain of virtual machine administration. Initial experiments show storage savings of up to 86% for some scenarios. These compression savings can enable new approaches to tasks such as machine image distribution and weekly virtual machine backups.

Our IZO compression prototype uses algorithms that originate from online deduplication filesystems. We find that these methods are well-suited for offline large-window compression as well. They require relatively little memory and little time when processing large data sets. While IZO removes global redundancies, it relies on small-window tools such as gzip to compress the remaining data. We observe that these two approaches are orthogonal to each other – the combined compression ratio is very close to the product of the individual compression ratios. An additional benefit is that the combined processing time of IZO+ gzip is generally less than the time it takes gzip to process the original input file alone. This is explained by the fact that IZO quickly removes global redundancies and gzip is left to compress a much smaller amount of data afterwards.

We are quite excited about the results of our initial experimental evaluation, but we also found several shortcomings of our prototype that we plan to address in our ongoing research.

First, we are concerned with the rapid growth of metadata in the backup scenario in Figure 11. Because the incremental changes between backups are small, we expect that the metadata between the two versions should be very similar as well. One way of confirming this would be to apply IZO to the archive yet again, but we'd rather store the metadata more efficiently in the first place.

Another area of interest is to determine optimal chunk size automatically. We currently require larger chunk sizes for larger input sets, because IZO eventually cannot keep all chunk hashes in memory, reducing the overall fold factor. It may be possible to begin compression with smaller chunks, and then increasing chunk size over time. A different approach that we would like to investigate is the encoding of superchunks – sequences of chunks that occur repeatedly because the underlying data is always contiguous as well (e.g., a large music file that is chunked into a sequence of hundreds of chunks). Lastly, we may be able to modify our hash eviction strategy to yield better results or lower memory requirements.

Finally, we are planning to add some convenience features to IZO, such as integration with gzip and bzip2, to increase the ease-of-use even further.

Acknowledgements

We wish to thank our shepherd, Derek Balling, for his guidance and comments, which improved our paper tremendously. Andrew Tridgell provided valuable insights into the algorithms and implementation of rzip. We would like to thank Simon Tsegay of the San Jose Eastside Boys & Girls club for the machine images used in the deep freeze experiments. Lastly, we are grateful to our anonymous reviewers for their comments and hope that the final version of our paper addresses their feedback.

Author Biographies

Mark A. Smith finished his graduate work at the University of California at San Diego with an M.S.-C.S., where he had worked in the Cardiac Mechanics Research Group on computer-aided visualization of heart mechanics. After graduation, he joined the Systems Research team at the IBM Almaden Research Center. Current areas of focus include filesystems, data deduplication, distributed security, and archival.

Jan Pieper received a Diploma Engineering Degree in Computer Science from the University of Applied Sciences in Hamburg, Germany in 2000, where he designed a web-based content management system. He joined IBM Almaden Research after graduation and has worked on distributed systems, web-scale data mining, file systems and social networking applications. He is currently also a part-time graduate student at UC Santa Cruz.

Daniel Gruhl is a research staff member at the IBM Almaden Research Center. He earned his Ph.D. in Electrical Engineering from the Massachusetts Institute of Technology in 2000, with thesis work on distributed text analytics systems. His interests include steganography (visual, audio, text and database), machine understanding, user modeling, distributed systems and very large scale text analytics.

Lucas Villa Real received a B.Ch.-C.S. in 2005 at the Unisinos University in Brazil, where he had worked with parallel computing and general purpose computation on GPUs. After graduation, he started his M.S.-C.S. program at the Polytechnic School at the University of Sao Paulo and joined their digital TV lab, focusing his research on demultiplexers and filesystems. In the meantime, he spent a few months doing an internship at the IBM Almaden Research Center, also working in filesystems.

Bibliography

[1] *Hyperfactor: A Breakthrough in Data Reduction Technology*, Technical report.

[2] Data Domain SISL, *Scalability Architecture*, Technical report, May, 2007.

[3] *Scalable, Highly Automated Virtual Tape Library Technology Reduces the Cost of Storing, Managing and Recovering Data*, Technical report, January, 2007.

[4] Tridgell, J. M. Andrew, *xdelta*, 2008, http://xdelta.org/ .

[5] Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 164-177, 2003.

[6] Clark, B., T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews, "Xen and the Art of Repeated Research," *USENIX Annual Technical Conference 2004, FREENIX Track*, pp. 135-144, 2004.

[7] Gilmore. J., *Gnu tar*, 2008, http://www.gnu.org/software/tar/ .

[8] Hitz, D., J. Lau, and M. Malcolm, "File System Design for an NFS File Server Appliance," *1994 Winter USENIX*, pp. 235-246, 1994.

[9] Gailly, J. and M. Adler, *gzip*, 2004, http://www.gzip.org/ .

[10] Kolivas, C., *lrzip*, 2008, http://ck.kolivas.org/apps/lrzip/README .

[11] Kulkarni, P., F. Douglis, J. LaVoie, and J. Tracey, "Redundancy Elimination Within Large Collections of Files," *USENIX 2004 Annual Technical Conference*, pp. 59-72, 2004.

[12] May, B., *NetApp A-SIS Deduplication Deployment and Implementation Guide*, Technical Report TR-3505, 2007.

[13] Park, K., S. Ihm, M. Bowman, and V. S. Pai, "Supporting Practical Content-Addressable Caching with czip Compression," *USENIX 2007 Annual Technical Conference*, pp. 185-198, USENIX, 2007.

[14] Policroniades, C. and I. Pratt, "Alternatives for Detecting Redundancy in Storage Systems Data," *Proceedings of the USENIX 2004 Annual Technical Conference*, pp. 73-86, 2004.

[15] Primmer, R. and C. D. Halluin, *Collision and Preimage Resistance of the Centera Content Address*, Technical report, 2005.

[16] Quinlan, S. and S. Dorward, "Venti: A New Approach to Archival Storage," *FAST 2002 Conference on File and Storage Technologies*, pp. 89-102, 2002.

[17] Rabin, M. O., *Fingerprinting by Random Polynomicals*, Technical Report TR-15-81, 1981.

[18] Reed, B., M. A. Smith, and D. Diklic, "Security Considerations When Designing a Distributed File System Using Object Storage Devices," *IEEE Security in Storage Workshop*, pp. 24-34, 2002.
[19] Rosenblum, M. and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *Computer*, Vol. 38, Num. 5, pp. 39-47, 2005.
[20] Seward, J., *bzip2*, 2007, http://www.bzip.org/ .
[21] Smith, J. E. and R. Nair, "The Architecture of Virtual Machines," *Computer*, Vol. 38, Num. 5, pp. 32-38, 2005.
[22] Sugerman, J., G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," *USENIX 2001 Annual Technical Conference, General Track*, pp. 1-14, 2001.
[23] Tang, J. C., C. Drews, M. Smith, F. Wu, A. Sue, and T. Lau, "Exploring Patterns of Social Commonality Among File Directories at Work," *CHI '07: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 951-960, 2007.
[24] Tridgell, A., *Efficient Algorithms for Sorting and Synchronization*. Ph.D. thesis, The Australian National University, 1999.
[25] Trigell, A., *rzip*, 2004, http://rzip.samba.org/ .
[26] Villanueva, B. and B. Cook, "Providing Students 24/7 Virtual Access and Hands-On Training Using VMware GSX Server," *SIGUCCS '05: Proceedings of the 33rd Annual ACM SIGUCCS Conference on User Services*, pp. 421-425, 2005.
[27] Waldspurger, C. A., "Memory Resource Management in VMware ESX Server," *OSDI '02: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pp. 181-194, 2002.
[28] Whitaker, A., R. S. Cox, M. Shaw, and S. D. Gribble, "Rethinking the Design of Virtual Machine Monitors," *Computer*, Vol. 38, Num. 5, pp. 57-62, 2005.
[29] You, L. and C. Karamanolis, "Evaluation of Efficient Archival Storage Techniques," *21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp. 227-232, 2004.
[30] You, L. L., K. T. Pollack, and D. D. Long, "Deep Store: An Archival Storage System Architecture," *21st International Conference on Data Engineering (ICDE'05)*, pp. 804-815, 2005.

Appendix: Birthday Paradox

The birthday paradox asks how many people need to be in a room together before the probability of two of them sharing a birthday is 50%. The formula to describe the probability is well known: $1 - n!/(n^c(n-c)!)$ where n is the namespace (365 days) and c are the number of things (people) with names (birthdays). In the case of the birthday paradox, there are 365 days, and by setting the probability to 50%, we arrive at the answer that it requires 23 people in a room before the likelihood of two of them sharing a birthday is 50%.

Portable Desktop Applications Based on P2P Transportation and Virtualization

Youhui Zhang, Xiaoling Wang, and Liang Hong – Tsinghua University, Beijing, China

ABSTRACT

Play-on-demand is usually regarded as a feasible access mode for web content (including streaming video, web pages and so on), web services and some Software-As-A-Service (SaaS) applications, but not for common desktop applications. This paper presents such a solution for Windows desktop-applications based on lightweight virtualization and network transportation technologies which allows a user to run her personalized software on any compatible computer across the Internet even though they do not exist on local disks of the host.

In our approach, the user's data and their configurations are stored on a portable USB device. At run time, the desktop applications are downloaded from the Internet and run in a lightweight virtualization environment in which some resource-accessing APIs, such as registry, files/ directories, environment variables, and the like, are intercepted and redirected to the portable device or network as needed. Because applications are "played" without installation, like streaming media, they can be called "streaming software." Moreover, to protect software vendors' rights, access control technologies are used to block any illegal access. In the current implementation, P2P transportation is used as the transport method. However, our design actually does not rely on P2P, and another data delivery mechanism, like a dedicated file server, could be employed instead to make the system more predictable.

This paper describes the design and technical details for this system, presents a demo application and evaluates it performance. The proposed solution is shown to be more efficient in performance and storage capacity than some of the existing solutions based on VM techniques.

Introduction

Today, a desktop PC is usually available at home, at work and at some public places. However, the diversified PCs, although running the same OS, cannot provide the user with her unique personalized desktop environment, which includes personal documents, frequently-used applications and their customizations.

Several solutions to this dilemma have been proposed. The first approach is thin-client computing [1]. CITRIX [2] is such a framework that allows a variety of remote computers to connect to a Windows NT terminal server to access a powerful desktop and its applications. And the remote computers only execute the graphical interface of the applications. For this solution, users' feeling would be bad when it is employed across the Internet, because of the long network latency.

Web-based application [3] is the second solution. In this mode a web browser is employed as a running platform for applications with the collaboration from a remote server. However, applications running on the platform are not compatible with the mainstream desktop environment: they should be rewritten for the web situation.

The third approach is the virtual machine-based solution. For example, IBM's SoulPad [4] carries an auto-configuring OS with a virtual machine monitor on a small USB device. The computer boots from the device and launches the virtual machine, thus giving the user access to her personal desktop. Moka5 [5], a commercial product, provides a similar solution. It uses the local machine's installed operating system (Windows XP or Vista) and runs the virtual machine (under VMPlayer) there. Recently, Moka5 also provided a network-based solution that users can download the VM image and use it on-the-fly. Collective [6] also propose a virtual-machine-based solution. It runs virtual machines on a PC and downloads OS images from Internet, rather than from the portable device.

The VM-based solution is promising because it is fully compatible with the mainstream desktop environment. However, virtual machine will introduce fairly substantial performance overheads. On one hand, as mentioned in [4], the VMWare-based configuration of SoulPad incurred a 26-29% increase in response time for Office Productivity and Internet Content Creation applications. And on the other, although the Xen virtualization environment is claimed to incurred only a 2-8% [7] slowdown in general owing to its paravirtualization architecture, it is necessary to modify the guest OS to take advantage of this feature. Now, some hardware-based virtualization acceleration can be used to deploy VM transparently, but this acceleration can introduce more overhead compared with the full software-based solution [8], especially for workloads that perform I/O, create processes, or switch contexts rapidly.

Moreover, carrying or downloading a whole VM-based OS is not economical. For example, a complete Windows XP installation occupies more than 1.5 GB disk space, which is too bulky for a tiny USB flash disk or the current Internet access rate.

It is believed that application virtualization will be the next frontier, and Software-As-A-Service (SaaS) is a promising deployment mode for software. Therefore, owing to the prevalence of portable storage devices and the ubiquitous network access, we propose a solution that combines the two technologies for any user to "play" her personalized software without installation anytime and anywhere conveniently. In this solution, the user's data and applications configurations are stored on a portable USB device. While at run-time, the desktop applications are streamed: downloaded from the Internet on-the-fly and run in an OS-level virtualization environment without installation.

In contrast to hardware-level virtual machine technologies, OS-level technologies have the virtualization layer positioned between the operating system and application programs. Every virtualization environment shares the same execution environment as the host machine, and only retains any divergences from the host as the VM's local state. Therefore, such an environment can have very small resource requirements and thus introduce only very limited overhead.

Under our approach, only personalized data and documents are stored on the portable device. Each personalized application runs in an OS-level virtualization environment that is layered on top of the local machine's Windows OS. At run-time, the virtualization environment intercepts some resource-accessing APIs, including those that access system registry, files/directories, environment variables, and the like, from these applications, and redirects them to those resources stored on the portable device or network as needed. For example, when one application accesses *My Documents, Desktop* or some other personalized configuration, it will reach the corresponding resources on the portable device instead of the local disk of the host machine.

In user's view, she can access her personalized applications and data conveniently on any compatible computer, even though they do not exist on local disks of the host. Moreover, because of the commonality of frequently-used applications, P2P transportation and the local look-aside cache can be used to improve the access performance.

Compared with the VM-based method, the storage capacity occupied by our solution is much smaller than other VM-based approaches, and the performance overhead introduced by virtualization is small. Of course, it relies on the host computer to provide the hardware resource as well as the compatible OS environment.

In addition, protecting software vendors' rights is a necessary prerequisite for a successful deployment mode. In our solution, only those processes running in the virtualization environment can have the right to access application files, and the user has to login a remote server before she launches the environment. Then, some mature billing mechanism can be employed, and illegal access will be prohibited.

In this paper, we first present the overview of our design, followed by implementation details, including how to run a software without installation, how to make software streaming in a friendly usage-mode based on file system filter and network transportation technologies, as well as the access control implementation. Then the prototype is introduced, and performance tests are presented. Finally, we present our conclusions, including comparison to of related works.

Design Philosophy

In our solution, a user can use her applications when plugging the portable storage into a Windows PC; and her personalized configurations, including *Desktop, My Documents, Favorites. Browser History, Temporary Internet Files, Cookies*. In addition, applications customizations (e.g., for a web browser, *Default Homepage, Internet Settings, Download Directory, Toolbars' Positions, Recently Opened Documents*) are restored just as if she were using her unique desktop environment on her home computer. Moreover, no trace of her work will be left behind on the host PC.

To provide these features, the following technical challenges should be overcome:

- How to run a Windows application without installation, including how to restore applications' customizations and user's personalized configurations transparently and how to leave no trace on the host PC.
- How to implement software streaming.
- How to enforce software licensing.

We will discuss each point separately below.

How to Run an Application Without Installation

Most Windows applications need to be installed before they can run normally. Even for an application that can work without installation, most of them save their customizations into the system registry and/or into configuration files located in some system folders. Then an application can be regarded as including two parts: Part 1 is all of the files and folders and registry keys and environment variables created by its installation process, and Part 2 is the customization produced during the run time.

To conquer the challenge, we have to make Part1 portable and enable the application to run in a sandbox where it can access and store the data associated with Part 2 in an isolation mode. The OS-level virtualization technologies are used to achieve these functions.

Therefore, this work consists of two tasks:

1) An installation snapshot, and
2) A runtime system design.

which are presented as follows.

Installation Snapshot

To make Part 1 portable, the modifications made by the application's installation process must be captured. There are usually two types of modifications: registry contents and files/folders. InstallWatch [9] is used to complete the task. It is a system monitoring tool that tracks changes to the computer's hard disk, registry, and .ini files when a new application is being installed.

In our implementation, a target application is installed on one clean Windows XP system. At the same time, InstallWatch is running to log those files created or modified in this process, as well as registry additions and modifications. Then, the files/folders created or updated are copied to a separated folder, called the *private folder*, while the directory hierarchy is retained. Similarly, the contents of the added/modified registry keys are collected to be stored in a separated file, which is called the *private registry file*.

Runtime System

Now we have captured Part 1 of one application, and the second issue is how to make it accessible by the application's executable file. *API Interception* is employed to do so.

API interception means to intercept calls from the application to the underlying running system and reinterpret the calls. It is usually used to extend existing OS and application functionality without modifications of the source code. Detours [10], a library developed by Microsoft Research Institute, are used to intercept those APIs employed to access the system registry and files/folders.

Owing to Detours, we have built *Wrapper APIs* that inject a wrapper DLL into the target process virtual address space as described in [10]. For example, when an application uses an interpreted WIN32 API to access the system registry, the wrapper API will be called firstly. Our injected code deals with this request before the original API – If one of the registry keys contained in Part 1 is to be accessed, the injected code can return the corresponding value from the *private registry*; otherwise the original API will be called to access the system resource. For requests for files and folders, a similar mechanism is adopted.

This makes the portability feasible because the customizations and files of an application have been isolated from the OS. And any modification happened during the run time will be store into the private registry or the private folder instead of the system's default position, while read operations are done wherever the content exists. Therefore no trace of application execution will be left behind on the host PC.

Streaming Software

Once we have succeeded in making the user-specific application state portable, the pivotal issue becomes where to locate the installation snapshot. Putting it on the portable device with the user's private data is not a bad idea. However, this approach disrupts traditional approaches to software licensing. The dominant licensing model for PC software permits use of the software "on a single computer," not "on a single USB drive." However, if they are placed on a network for downloading on demand, some mature billing mechanisms, like those employed in SaaS, can be used to protect vendors' rights.

Therefore, in the current implementation, applications are stored on the Internet and downloaded on demand while users' private data is still kept on the portable device for privacy.

However, it can be difficult for ordinary users to run applications transparently and quickly when streaming software is used. Thus, we have implemented a solution based on file system filter driver [11] technology.

A file system filter driver intercepts requests targeted at a file system. By intercepting the request before it reaches its nominal target, the filter driver can extend or replace functionality provided by the original target. Owing to this feature, a file system virtualization mechanism, which we call an *anchor file*, is achieved to present users a friendly interface.

For example, when the user launches an executable file, `z:\abc.exe`, the shell program will send a serial of IRPs (I/O Request Packets) to the file module of OS, which is intercepted by our filter driver. Then the driver will deal with these IRPs and return results directly rather than transfer those to the target. So, while it looks like the `z:\abc.exe` is being accessed, in fact the filter driver has transformed and redirected these requests to an Internet location. `z:\abc.exe` is just an *anchor*.

Those file system visits generated while the application is running will be handled in a similar way. In other words, our filter driver will judge whether the installation snapshot should be accessed or not. If the answer is yes, the drive will redirect the requests to the remote location. Moreover, because of the commonality of frequently-used applications, in the current implementation P2P transportation and look-aside cache are used to improve the access performance. More details about the virtualization approach and4 software streaming are presented in the next section.

Access Control

Copyright violation is a real problem that hampers the software industry's progress. In this play-on-demand mode, it is especially important to prevent any illegal access; otherwise, running software without installation will be good news for illegal users.

Therefore, before the user launches our own shell program with the file system filter, she has to identify herself and login to the server.

Moreover, as mentioned previously, users can access the application files just like they are using the local file system, so how to prevent the illegal download is another key consideration. Otherwise, a user can copy applications to her local disk and use them without login.

In our solution, an access control mechanism is implemented to protect some essential files (such as the executable and DLL files of applications). It works by allowing only certain processes in the virtualization environment (like our own shell program) to access those files. If the user attempts to use another program (for example, explorer.exe) to copy one essential file out, our filter driver will intercept those IRPs to identify whether they are issued from a legal process or not. Because explorer.exe is a program outside of the virtualization environment, its access will be denied.

Implementation

Virtualization Running Environment

Because there are some existing similar OS-level implementations, like Progressive Deployment System (PDS) [12] and Featherweight Virtual Machine (FVM) [13], only a brief overview of our implementation is presented here.

The captured installation snapshot can be divided into six categories:

- Added registry set. It contains the entries created by the installation.
- Modified registry set. It contains the entries whose values or sub-keys have been modified or deleted.
- Deleted registry set. Those entries deleted by the installation are included. So that the entries in this set will not be accessed during the run time.
- Added file set. It is similar to the added registry set, including new files and new folders created by the installation.
- Deleted file set. It is similar to the deleted registry set.
- Modified folder set. For any file or folder in the added/deleted file set, its parent folder will be included in this set.

These six sets are not fixed. They may be modified when the application runs.

The private registry is a complete registry system that provides access APIs just as Windows OS does. It works like a small subset of WINE [14], which is an open source implementation of the Windows API on top of UNIX. In other words, our private registry provides another implementation of registry APIs.

When the target application is launched, the six sets and registry contents will be initialized. The absolute path is used to identify a single registry key and such a map structure is maintained, which can map a handle of any opened key to its full path. For example, when one program opens the registry key *"HKCR*.doc" the interception code will map the returned handle to the path string. Then every time this handle is used, its full path can be referred to.

The registry-related APIs below have been wrapped.

Open/Create a Key (RegOpenKeyEx / RegCreateKeyEx)

Arguments of RegOpenKeyEx contain the handle of an opened key and a null-terminated string indicating the name of the subkey to open. Based on the above-mentioned map, we can identify the absolute path name for the key to open, and thus find out which set it belongs to. If the key is in the added registry set, it will be opened in the private registry; if it is in the modified registry set, it will be opened in both the private registry and the system registry. The two handles referring to both keys are stored in another handle-map structure that will be used in subsequent invocations, and the system handle is returned to the application. If the key is in the deleted registry set, this invocation fails. If the key is not present in any of the registry sets, then the original system API will be used.

Creating a key is often regarded as an open operation except that it will create a new key when the key does not exist. In this case, the new key is created in the private registry and its full path is inserted into the added registry set. In addition, its parent key is moved into the modified set.

Set Value of a Key (RegSetValueEx)

Any new value is always saved in the private registry, and its parent key will be moved to the modified set (the exception is that it has been in the added set).

Query/Enum Value of a Key (RegQueryValueEx / RegEnumKeyEx / RegEnumValue / RegQueryInfoKey)

If the key is in the added set, it will be queried only in the private space. Similarly, the query will be done in the system registry if the key does not exist in any predefined set. The most complicated case occurs when the key is in the modified set. Both of the private and the system registries should be queried, and the results will be merged before return. It means, if there is any duplicated key, the private one should be presented instead of the system one. The same method is adopted for accesses to key values.

Close Key. (RegClosekey)

When closed, the key's corresponding handles must be released and removed from both maps.

Delete Key/Value (RegDeleteKey / RegDeleteValue)

If the key is in the added or the modified set, it will be deleted from the private registry and inserted into the deleted registry set. If it exists in the system registry, it will not be actually deleted. Instead we add it into the deleted registry set. For subsequent accesses later, we first check whether the key is in the deleted registry set; if so, nothing is done but an error code is returned. Deleting a value is handled in the analogous way.

In summary, the principle is that any modification is always saved in the private space while any query will return the combination of results from both registries. In addition, if there is any duplication, the private registry has the higher priority.

Since the *private folder* is located in a portable device whose drive letter will change on different hosts, the registry value containing such a full path is modified to represent the current position before return.

For APIs that access the file system, a similar method is adopted because folders can be regarded as registry keys and files can be regarded as values. Moreover, the copy on write (COW) mechanism is applied at the whole file level when a file is modified during run time.

For environment variables, the solution is much simpler. As we know, a process will inherit environment variables from its parent process. In our implementation, a shell program is in charge of launching any target application. Therefore, it can set any application-specific variable before launching the target.

Streaming Software Based on File System Filter Driver

In order to explain the implementation clearly, we begin with an example operation.

The user inserts a USB device into the Windows host and then our own shell program (with the filter driver) is loaded automatically by an auto-run mechanism. Thereafter, any file system I/O operation will be intercepted.

The portable application is located on the USB device, in z:\program files\abc\. But in fact the physical position of this folder is empty and the real application is stored on a remote file server.Our shell program presents a shortcut of this application to the user.

When the user clicks the shortcut to launch the application, the shell program will send some IRPs, including IRP_MJ_CREATE, IRP_MJ_DIRECTORY_CONTROL, IRP_MJ_READ, IRP_MJ_CLEANUP and IRP_MJ_CLOSE.

IRP_MJ_CREATE is used to open the folder/file and then one or more IRP_MJ_DIRECTORY_CONTROL IRPs are issued to query the folder/file information, followed by lots of IRP_MJ_READs to read the real data. The last two IRPs end the operation series.

During this process, our filter driver handles all IRPs issued to this folder/file entirely. Specifically, it simulates the target to return folder/file information (metadata) and data that are obtained from the network server in reality. So, in the shell's – and therefore the user's – view, this application can be launched as a local one.

Work Flow. During the start-up stage, our shell program connects to the remote server to download all metadata of the anchor files and folders. Because this file system does not exist physically on the portable device, these metadata must be retrieved first for presentation. Fortunately, their total size is very small, which only delays the startup time for a few seconds.

All metadata are transferred to the filter driver, which had learned previously which folders/files are to be managed and what should be returned when they are accessed.

As it runs, the portable application will issue many IRPs and which will then be intercepted by this driver. If an IRP is issued for metadata, it will be handled by the driver itself. IRPs for file data will be retransferred to another user-level module, the *client module*, which will download the data from the network.

Details of IRP handlers. Because no Fast I/O is supported in our driver, any I/O request will generate IRP operations. The IRPs below are intercepted (for more info for IRP and Fast I/O, please refer to [11]).

CREATE IRP

When any file or folder is to be accessed, this is always the first IRP. In its dispatch function, the full path of the target is obtained using the *ObQueryNameString* API at first. If it is an object belonging to the install snapshot, the target's file object address will be stored into an internal hash table. We know all subsequent IRPs will carry their object addresses, so this table can be used to check whether an IRP should be handled by our dispatch functions or by the system default route.

DIRECTORY CONTROL IRP

This IRP contains two subtypes: IRP_MN_NOTIFY_CHANGE_DIRECTORY and IRP_MN_QUERY_DIRECTORY. The first one can be skipped in our solution. The second is used to get information about all files and subfolders of the target directory. The corresponding metadata obtained at the startup stage will be returned.

QUERY INFO IRP

It is used to query file metadata and can be handled like the preceding one.

READ IRP

Its dispatch function will put the IRP into its internal waiting queue and simultaneously signal a waiting event to notify the client module that there is some request to deal with. The module then reads the request using the *DeviceIoControl* API and downloads the requested data from the network. After downloading, it will call *DeviceIoControl* API to send the data to the driver. On receipt of the data, the filter completes that pending IRP.

In the current implementation, the driver waiting queue can contain up to 5000 requests, which is enough for most common use cases.

CLEANUP/CLOSE IRPs

This pair of IRPs always appears as the end of a series of operations to one target. They remove the responding target's entry from the hash table when present. At the same time, its dispatch function will notify the client module that the access series finished.

P2P Transportation and Optimization. The portable application can be downloaded through a single dedicated server or it can be downloaded from multiple servers in parallel to speed up the process. Since the implementation of a dedicated server is rather straightforward, we use libtorrent [15], an open source library that implements BitTorrent [16] protocol, to achieve P2P transportation. Because P2P is a mature technology, its technical details are skipped here.

In addition to speed up the access performance, the client module creates a look-aside cache on the USB device. We know that in BitTorrent a downloadable file is divided into many fixed-length pieces, with each piece indexed by its hash value. So, we use a content-based addressing scheme, similar to several previously discussed in the literature [17] to implement a nonvolatile cache. When a particular piece is accessed, the local cache will be first, avoiding a network operation if the data is already present. When the virtual environment exits, all cached data will be stored for the next time.

The content-based addressing scheme brings us three other benefits:

1. It can reduce the storage requirements because content-addressing storage is also a good compression mechanism.
2. Store files in this mode rather than storing them directly can prevent users from recovering application files from the cache.
3. Hash value-based indexing allows any tampered content to be detected easily.

In addition, some pre-fetch technology is employed. Specifically, when one piece of a file is requested, the whole file or even the whole application to which that file belongs will begin to be downloaded. Obviously this pre-fetch can be highly efficient.

Finally, the client machines are able to serve out the applications themselves once they have them cached, which is why it is called P2P.

So far, we have discussed only read operations. So how does our approach handle write operations?

We know that most snapshot files are read-only; only configuration files (including *private registry files* mentioned earlier) will ever be modified. For these files, the copy-on-write strategy is employed: when a file is modified, it will be downloaded completely to the local USB device, and any subsequent modification will happen locally. Note that there is no coherency problem to worry about since modified files belong to users' personal customizations.

Access Control Based on WMI Service

Windows Management Instrumentation (WMI) [18] is the infrastructure for management data and operations on Windows-based operating systems. It can be used to get information of all current processes (using *CreateToolhelp32Snapshot* API) and can send corresponding notifications when any process is created or terminates (using ExecNotificationQuery Async API).

Based on these features, the client module creates the process-hierarchy structure at startup. During the application's running time, any new process's ID and their parents can be detected and captured. In this way, a whole process hierarchy can be maintained in real time.

The access control policy is based on process ID. In our design, the root of the hierarchy is the process ID of our shell program, which can access all installation snapshot files while any processes outside of the hierarchy is forbidden. Furthermore, a process and its offspring can only access the files belonging to its own application. Therefore, when the client module accepts a read request from the filter driver, it will determine whether the original sender is an authorized one or not based on the process hierarchy. Only legal requests are allowed.

The Prototype

Overview

A prototype implementation of our solution has been completed using VC 7.0. And many existing applications can be made portable, including *MS Word 2003, MS Excel 2003, MS PowerPoint 2003, Lotus Notes, Photoshop, Internet Explorer 6.0, Outlook Express 6, Winzip, UltraEdit, FlashGet, Bittorrent* and so on.

Of course, there are still some applications that cannot currently be made portable using our approach. For example, applications that have their own kernel modules are not supported because our solution only supports access- redirection for user-level resources. Another problem is that the current implementation does not consider Windows Side by Side technology (SxS) [19] and so some new applications like Microsoft Office 2007 and Adobe Reader 8 are not supported. With SxS technology, applications can install DLLs to version-specific directories and tell Windows what version of the DLL should be used when they load a DLL by that name. However, we plan to support this feature in the next version.

The whole work flow of the prototype is described in Figure 1. Initially, installation snapshots of applications are captured by the server and are stored on the data server, which also functions as the tracker server for BitTorrent P2P transportation. When the application runs, the virtualization environment is running on multiple clients that are connected with each other for P2P data sharing.

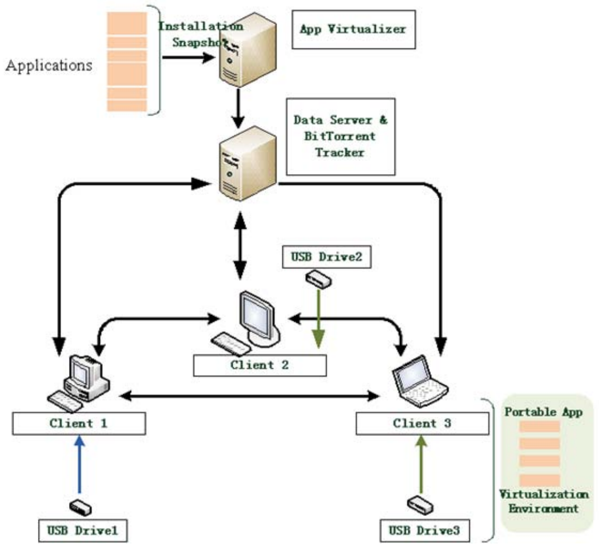


Figure 1: The work flow.

The shell program of this prototype is a *start-menu*-like GUI where the user can launch any portable application as she operates the system's Start menu. Of course, these applications are located on the portable device instead of being installed on the host.

Besides portable applications, most personal folders such as *My documents, My Desktop, Cache paths, Temporary Internet Files*, etc., are also portable. For example, when Outlook Express is launched from our prototype, its email boxes are located on the portable device, and other customizations, including the account info and the email signature, etc. are personalized. Moreover, all newly received emails, as well as modifications of its customization settings, are stored onto the portable device so that the program is enabled to be portable. In contrast, when Outlook Express starts from the system's start menu, all of its configuration settings belong to the owner of the host. Once the portable device is unplugged, no trace of work is left behind on the host PC.

In addition, there are some visual indications to users that they are running a portable application. For example, when the portable Outlook Express is launched, its title will be modified to "Outlook Express – Desktop2Go" by intercepting the *CreateWindow* API, reminding the user not to save newly created files to the local hard disk.

When the portal application exits, all modifications to the system registry and the file system are stored on the portable device. Therefore, when the application is launched again from a different computer, the user's latest modifications are present.

Additional Technical Details

Some Windows pre-installed applications, such as IE and Outlook Express, are deeply integrated into the OS, making it very difficult to separate them from the operating system. On the other hand, this also means that they are always available on a compatible host. For such applications, only their customizations and personalized data are made portable.

For example, when IE (located on the host system) is launched from our GUI, it will run in the virtualization environment and its registry APIs are intercepted. Then, when it accesses registry entries that store customizations, like *Home Page, Download Folder, Favorites, Browser History, Internet Temporary Files* and so on, the interception code will return values from the private registry so that the portable personalized customizations are implemented.

For registry entries, our principle is that any modification is always saved in the private space while any query will return the combination of results from both registries. In reality, lots of registry accesses can be skipped. Most applications do not write their implementation from stem to stern. Instead, many standard Windows components will be employed. For example, when an application shows an *open file* dialog, many registry accesses occur, but they are totally transparent to the application since they are invoked by system code. Therefore, registry accesses can be divided into two categories: application-specific ones that program developers complete intentionally and ones generated from system components. The latter can be ignored by the virtualization system.

For instance, Adobe Reader creates the key *"HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Acrobat Reader"* to save its configurations. Therefore, only those entries below this key handled in the manner described above, and all other registry accesses are left to the host system. Of course, differentiating the two types of access depends on the specific application. Fortunately, for Windows OS, some public publications, like [20], have explained which registry entries are system-related.

Our last point is about shell folders mapping. *TEMP, HOMEPATH, USERPROFILE, APPDATA*, and other system directories are called shell folders in Windows. Applications typically use them and consult the system registry in order to locate the current user's My Documents, Desktop, and other personalized folders. Because these folders are also portable, accesses to the related registry entries have to be redirected into the private space. Specifically, the following registry entries are also stored in the private registry:

- *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders*
- *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders*

- *HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders*

In this way, portable applications can transparently visit personalized folders located on the mobile device.

Performance Testing

Application response times are the key metric of our prototype's usability. The time it takes for applications to respond to user-initiated operations is a measure of what it feels like to use the system for everyday work.

The test platform is a Windows XP SP2 PC, equipped with 1.25 GB DDR SDRAM, one Intel Pentium 1.6 GHz CPU and the 100 Mb Ethernet. The hard disk is one Hitachi IC25N030ATMR04 Travelstar.

The Common Desktop Application (CDA) benchmark is used. It automates the execution of common desktop applications of the Microsoft Office suite and IE browser in the Windows XP environment. During this process, nine local documents (including three Word files, three Excel sheets and three PowerPoint presentations) are opened, edited and closed automatically, while three instances of IE are running as the background load. Finally, all applications are closed and the running time is logged.

In all test cases, the pre-fetch mechanism is enabled.

Test Case 1: Perfect Local Cache

In this case, the look-aside cache performance is assumed to be perfect: all reads are cache hits.

System Configuration	Workload	Normalization Value
Physical, IDE	39.6 s	1.000
Virtual, IDE	43.2 s	1.091
Virtual, USB	47.9 s	1.210

Table 1: Response times.

Table 1 shows the running time averages across three runs of the benchmark. The row labeled "Physical, IDE" represents our baseline configuration, applications running on the physical machine and from the internal IDE drive, meaning our environment is not deployed.

The row labeled "Virtual, IDE" corresponds to running applications in the virtualization environment, with the cache located on the internal IDE drive. It is intended to isolate the overhead of virtualization from the overhead of using an external drive.

The row labeled "Virtual, USB" corresponds to locating the cache on the PocketDrive connected via USB 2.0.

The results show that moving to a VM-based configuration with the cache on the IDE drive incurred a 9.1% increase in response time. Moving to a VM-based configuration with the cache on the USB drive incurred a 21.0% increase over the baseline. So, the overhead introduced by virtualization (including the user-level virtualization and filter driver) is much smaller than that due to using external storage for the cache.

Of course, perfect cache performance is not the common case. However, if the user always uses some limited portable applications and if the cache space is large enough, we believe the cache will behave very nicely.

Test Case 2: No Local Cache

For this case, our testing client is located in the China Education & Research Network (CERN) while the BitTorrent tracker server is placed in an IDC outside of CERN; and a P2P transportation environment is constructed. For more information about this infrastructure, please refer to [16]. In this case, the local cache is disabled. But this does not mean that there is no cache in the whole system: the default file cache of OS is still present.

We can identify three subsets of the runs for this test case:

- **One Remote Peer**. In this case, one remote peer (located outside of CERN) is running, which holds all portable applications. So, as the testing client starts to access files, it can download them from this peer as well as the tracker server.
- **One Local Peer**. In this case, one local peer (located in the same 100 Mb Ethernet-LAN as the testing client) is running, which holds all portable applications. So, as the testing client starts to access files, it can download them from this peer as well as the tracker server.
- **More Than One Peer**. In this case, two, four, or eight local peers (located in the same LAN as the testing client) are running, and each of them holds all portable applications. So, as the testing client starts to access files, it can download them from these peers as well as the tracker server.

The test results show that when there are only limited, remote peers, the system performance is fairly slow compared with the baseline. However, we believe in reality the situation will be not so pessimistic:

- First, in our observation, only the startup process of an application shows slowdown, and when the application's GUI is presented, the operation delay is acceptable. Because it is difficult to accurately define what the startup time is and we cannot separate it from the whole operation time, we can only present our subjective feelings here.
- Secondly, because most users access only a limited number of frequently-used applications, we can expect the local cache to exhibit a high hit ratio.

- Finally, in our tests human response time was not emulated. However, in reality, the natural delays that occur when people are working are helpful to hide the background download latency.

System Configuration	Workload	Normalization Value
Physical, IDE	39.6 s	1.000
One Remote Peer	201.9 s	5.098
One Local Peer	82.3 s	2.078
Two Local Peers	65.3 s	1.649
Four Local Peers	52.2 s	1.318
Eight Local Peers	50.9 s	1.285

Table 2: Response time.

Discussion

Software Licensing

Virtualization disrupts traditional approaches to software licensing. For instance, the arguably dominant licensing model for PC software permits use of the software "on a single computer." Is it necessary to modify it to "for one person"?

Alternatively, we can extend this solution to an Internet-based service, where the private space will be located in a network server and accessible to every connected computer. Then, any access to the applications can be logged by the service-provider, which can be used for accounting.

Security and Privacy

Our solution does not write any customization to storage on the host PC. This isolation can keep the OS pristine, helping prevent and contain security breaches and infections.

Unlike some VM-based methods, this solution works at OS-level and highly depends on the host OS, so user should trust the host before operating on it. On the other hand, our VM-based solution is safer in some ways than using an unknown PC since it does not run any software previously installed on the host and starts the host from a known power-down state. However, if the local BIOS is compromised, it is equally unsafe as using the host directly.

Another open question is whether our solution is safe or not if the host has already been infected by some virus. Because our access control mechanism can prevent any illegal access to the protected application files, it seems that the virus can not impair them.

Finally, we believe it is necessary to construct a trusted chain between the user and the host to solve this problem completely, which depends on the prevalence and availability of trusted computing.

After these problems are solved, we can expect that software-on-demand may become one important deployment model for common personal applications. Just like today's video-on-demand (VOD) services, maybe some SOD (software-on-demand) providers will spring up, and a user can play her personalized software on any compatible hosts without installation and without conflicts. Some applications can even be subscribed to and pushed to one's host, so the download latency is hidden.

Others

In the current implementation, P2P transportation is used as an architectural requirement. In fact, it is apparent that other data delivery mechanisms can be employed. For example, to deploy such a system in an enterprise, one or more central servers can be used as the application source.

Another adaptation is that, if the host has the same application and its components, our system should use these components instead of downloading them from the Internet. But in this situation, if the host application is infected with a virus/malware, other applications running inside VM are vulnerable to security issues (even if we assume that our access control mechanism is not infected). A solution to this problem can be addressed by having the component be verified based on its hash value before it is used. If the verification fails, the component is downloaded even though it is present locally.

Related Work

Portable Systems & Software

Chen and Noble [21] observed that virtual machine technology [22] can be used to migrate sessions between computers and thus be used for mobility. Internet Suspend/Resume [23] demonstrated that using commercial VM technology such as VMware Workstation, together with a networked file system such as Coda [24], makes it possible to walk up to a machine and resume a suspended session. Each ISR client has a Host OS and VMware Workstation preinstalled and has access to a networked store of VM images.

SoulPad is a new approach based on carrying an auto-configuring operating system along with a suspended virtual machine on a small portable device. With this approach, the computer boots from the device and resumes the virtual machine, thus giving the user access to his personal environment, including previously running computations. SoulPad has minimal infrastructure requirements and is therefore applicable to a wide range of conditions, particularly in developing countries.

A Soulpad-like solution is DoK (Desktop on Keychain) [25]. In contrast, it uses the local machine's installed operating system (Windows), and runs VirtualPC there, resuming the user's previous virtual machine session.

A commercialized product is Moka5 LivePC, which contains everything needed to run a virtual computer: an operating system and a set of applications.

LivePCs can be used on the desktop, or users can take them on a portable USB drive or access them through a network server. In the latter case, a central difference server can provide a VM image "diff" through streaming, so that the end user will not experience undue delays on system startup.

Unlike these approaches, our solution is based on ultralightweight virtualization. Accordingly, the storage capacity required by our solution is much smaller, and the performance overhead introduced by virtualization is almost negligible.

Some recent commercial offerings attempt to support personalization of anonymous PCs at OS-level. For example, Migo [26] allows users to carry personal settings and files on a USB flash key. One limitation of this approach is that it must be tailored for each application to be migrated. Moreover, it only saves personalized data into the USB storage, not the applications themselves.

U3 [27] presents a development specification of portable applications, which means software should be rewritten for portability. So compared with our solution, it is not a transparent one.

A similar solution is Ceedo [28], but there is no documentation of its implementation technologies.

Software As Service (SaaS)

SaaS is a software usage mode where the vendor develops a web-native software application and hosts and operates the application for use by its customers over the Internet. In the past, this mode was usually used by some enterprise-level software, including CRM, SCM, ERP and so on. Now, it has moved closer to ordinary users.

One example of this approach are web-based applications such as Google Docs & Spreadsheets [29], in which a web browser is usually employed as a running platform for word processing and spreadsheet applications. This looks like a promising software delivery model; however, the applications have to be rewritten for the Internet environment. So for the existing desktop applications, a compatible model is preferred.

Virtualization has also been deployed in on-demand software. One solution is Progressive Deployment System (PDS), which is a virtual execution environment and infrastructure designed specifically for deploying software on demand while enabling management from a central location. PDS intercepts a selected subset of system calls on the target machine to provide a partial virtualization at the operating system level. This enables software's install-time environment to be reproduced virtually while otherwise not isolating the software from peer applications on the target machine.

Another similar and practical solution is Microsoft's SoftGrid [30], which can help IT departments to cut costs while increasing operational agility and reducing conflicts. SoftGrid can convert applications into virtual services that are managed and hosted centrally but run on demand locally. Application virtualization reduces the complexity and labor involved in deploying, updating, and managing applications.

However, both PDS and SoftGrid are designed for LAN environments and are not general solutions.

Our solution is an interesting and helpful exploration for deploying personal software on demand in a compatible way. Compared with other works, it combines application virtualization and SaaS technologies together: users can access their personalized applications over the Internet with some access control, which is speeded up by P2P transportation and the local cache.

Others

Using P2P to improve system's start-up performance has been used in some other systems, like Moobi [31]. Moobi uses BitTorrent to provide efficient distribution of the image cache of a dataless-workstation system, and it can support many more nodes to startup simultaneously compared to conventional network booting.

Another work we have referred to is OS Circular [32]. It is a framework for Internet Disk Image Distribution of software for virtual machines, those which offer a virtualized common PC environment on any PC. Especially, it used FUSE [33], a user-space file system framework, to implement a stackable virtual disk across the Internet, which inspired us to design the anchor file system.

Summary and Future Work

This paper presents a solution for portable Windows applications/customizations based on OS-level virtualization and P2P technologies. This solution can separate an application's private files/folders/registry entries into a portable device and employs *API interception* to make the application transparently access these resources at run time. From the user's viewpoint, she is able to access her personalized applications and data conveniently on any compatible computer, even though they are not present on the local disks of the host computer.

In addition, to present a friendly interface for users, file system virtualization is implemented, combined with access control to prevent any illegal software usage. P2P and local caching are used to speed up the download performance.

The design principle is presented and a prototype solution is introduced. We find in reality some practical issues, such as pre-installed applications, shell folders and application-irrelevant accesses, should be dealt with individually. Owing to the lightweight virtualization, the extra performance overhead mainly comes from the network latency, which can be decreased by P2P transportation and the local cache.

We believe this solution can be used not only for personal computing across the Internet but also for software deployment and administration in enterprises. In this mode, software installation can be converted into content distribution, which is hosted centrally but runs on demand locally. Doing so will reduce the PC total cost of ownership (TCO) significantly.

Currently we put users' data and customizations on the portable device for privacy. In the future, we plan to discard the device entirely and employ the network to provide and store everything. The challenge lies in how to keep coherence of personal data across the Internet. Moreover, we intend to address the problem of upgrading portable applications transparently, especially upgrade those cached on portable devices, in the next version. Some existing Content Distribution Network (CDN) technologies will be referred to.

Another issue with the current version is that the filter driver-based implementation requires the user have Administrator' access, which violates the principle of least privilege [34]. We plan to achieve a user-level version in the next version.

Author Biographies

Zhang, Youhui is an Associate Professor in the Department of Computer Science at the University of Tsinghua, China. His research interests include portable computing, network storage and microprocessor architecture. He received his Ph.D. degree in Computer Science from the same university in 2002.

Wang, Xiaoling is a Master's degree student in the Research Institute of Information Technologies at the University of Tsinghua, China. Her current research interests include portable computing and computer system simulation.

Hong, Liang is a Master's degree student at the Beijing University of Post and Telecommunication, China. Currently he is doing his research work at the University of Tsinghua, and his research field is portable computing.

Acknowledgement

We would like to thank Prof. Brent B. Hoon Kang, and Ms. Aeleen Frisch and our anonymous reviewers for their time and insightful comments regarding this paper.

The work is supported by the High Technology Research and Development Program of China under Grant No. 2006AA01Z111 and the National Grand Fundamental Research 973 Program of China under Grant No. 2007CB310900.

Bibliography

[1] Richardson, T., Q. Stafford-Fraser, K. R. Wood, et al., "A Virtual Network Computing," *Internet Computing, IEEE*, Vol. 2, Num. 1, January, 1998.

[2] *CITRIX*, http://www.citrix.com .

[3] http://en.wikipedia.org/wiki/Web_application .

[4] Caceres, Ramon, Casey Carter, Chandra Narayanaswami and Mandayam Raghunath, "Reincarnating PCs with Portable SoulPads," *Proceedings of the Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005)*, June, 2005.

[5] http://www.moka5.com/ .

[6] Chandra, R., N. Zeldovich, C. Sapuntzakis, and M. S. Lam, "The Collective: A Cache-Based System Management Architecture," *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI 2005)*, May, 2005.

[7] http://www.cl.cam.ac.uk/research/srg/netos/xen/performance.html .

[8] Adams, Keith and Ole Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, March, 2006.

[9] http://tejasconsulting.com/open-testware/feature/installwatch.html .

[10] Hunt, Galen and Doug Brubacher, "Detours: Binary Interception of Win32 Functions," *Proceedings of the Third USENIX Windows NT Symposium*, July, 1999.

[11] *File System Filter Drivers*, http://www.microsoft.com/whdc/driver/filterdrv/default.mspx .

[12] Alpern, Bowen, Joshua Auerbach, et al., "PDS: A Virtual Execution Environment for Software Deployment," *Proceedings of the First ACM/USENIX International Conference on Virtual Execution Environments*, March, 2005.

[13] Yu, Yang, Fanglu Guo, Susanta Nanda, Lap-chung Lam and Tzi-cker Chiueh, "A Featherweight Virtual Machine for Windows Applications," *Proceedings of the Second ACM/USENIX Conference on Virtual Execution Environments (VEE'06)*, June, 2006.

[14] http://www.winehq.org/site/docs/wineusr-guide/index .

[15] http://sourceforge.net/projects/libtorrent/ .

[16] Cohen, Bram, "Incentives Build Robustness in BitTorrent," *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June, 2003.

[17] Morrey III, Charles B. and Dirk Grunwald, "Content-Based Block Caching," *Proceedings of 23rd IEEE Conference on Mass Storage Systems and Technologies*, May, 2006.

[18] *Windows Management Instrumentation*, http://msdn.microsoft.com/en-us/library/aa394582.aspx .

[19] http://en.wikipedia.org/wiki/Features_new_to_Windows_XP .

[20] Russinovich, Mark E. and David A. Solomon, *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000*, Microsoft Press, January, 2005.

[21] Chen, P. M. and B. D. Noble, "When Virtual is Better Than Real," *Proceedings of IEEE 8th Workshop on Hot Topics in Operating Systems*, May, 2001.

[22] Goldberg, R. P., "Survey of Virtual Machine Research," *IEEE Computer*, Vol. 7, Num. 6, June, 1974.

[23] Kozuch, M. and M. Satyanarayanan, "Internet Suspend/Resume," *Proceedings of 4th IEEE Workshop on Mobile Computing Systems and Applications*, June, 2002.

[24] Satyanarayanan, M., "The Evolution of Coda," *ACM TOCS*, Vol. 20, Num. 2, May, 2002.

[25] Annamalai, Muthukarrupan, Andrew Birrell, Dennis Fetterly and Ted Wobber, "Implementing Portable Desktops: A New Option and Comparisons," *Microsoft Research (MSR)-2006-151*, October, 2006.

[26] http://www.migosoftware.com/default.php .

[27] http://www.u3.com .

[28] http://www.ceedo.com/ .

[29] http://documents.google.com/ .

[30] http://www.microsoft.com/systemcenter/softgrid/default.mspx .

[31] McEniry, Chris, "Moobi: A Thin Server Management System Using BitTorrent," *Proceedings of 21st Large Installation System Administration Conference*, November, 2007.

[32] Suzaki, Kuniyasu, Toshiki Yagi, Kengo Iijima, and Nguyen Anh Quynh, "OS Circular: Internet Client for Reference," *Proceedings of 21st Large Installation System Administration Conference*, November, 2007.

[33] *Filesystem in Userspace*, http://fuse.sourceforge.net/ .

[34] http://en.wikipedia.org/wiki/Principle_of_least_privilege .

Topnet: A Network-Aware top(1)

Antonis Theocharides, Demetres Antoniades, Michalis Polychronakis, Elias Athanasopoulos, and Evangelos P. Markatos – Foundation for Research and Technology; Hellas, Greece[1]

ABSTRACT

System administrators regularly use the top utility for understanding the resource consumption of the processes running on UNIX computers. Top provides an accurate and real-time display of the computing and memory capacity of the system among the running processes, but it provides no information about the network traffic sent and received by the processes running on the system.

Although we've seen a proliferation of network monitoring tools that help system administrators understand the traffic flowing through their networks, most of these tools have been designed for network deployment and can not easily, if at all, provide real-time attribution of network resources to individual processes running on end hosts.

In this paper, we describe the design and implementation of *Topnet*, an extension of the top UNIX utility that provides a *process-centric* approach to traffic monitoring. *Topnet* presents users with an intuitive real-time attribution of network resources to individual processes. Our evaluation suggests that *Topnet* through (i) the familiar user interface of top and (ii) a reasonable performance overhead, provides an accurate way to attribute network traffic to individual processes, enabling users to have a more comprehensive process-aware understanding of network resource consumption in their systems.

Introduction

The UNIX top utility [34] is used daily by users and system administrators for real time monitoring of system and process information. Top provides a continuously updated breakdown of system resources such as CPU and memory of the running processes, as well as process-specific information such as the state, priority, size, total CPU time, and so on. However, top does not provide any information about how the network resources of the system are utilized by the different processes. Although there exist several system utilities that display information about the network configuration, connection state, or statistics per network interface, currently there is no convenient way to attribute the system's network traffic to the individual processes that send or receive it.

With the proliferation of network services and systems, it is increasingly difficult for users to keep track of which applications communicate through the network, when this is happening, and how much traffic each application sends and receives. For example, besides the operating system, popular applications such as browsers, media players, and productivity suites, among many others, communicate with remote servers for automatic updates. In many cases the user is unaware of this activity. Similarly, the network activity of peer-to-peer software like file sharing or video conferencing applications is not always directly related to the user's actions and thus the exact network behavior is unclear to the end user.

Consider, for example, a Voice-over-IP (VoIP) application, such as Skype. Even when the user is not directly using the application, Skype may still consume processor and network resources to relay other users' traffic and keep the overlay network well connected. Thus, in the presence of such applications generating network traffic in the background, administrators would like to be able (i) to understand when such traffic is generated, and (ii) to pinpoint the processes that are responsible for the generation of this traffic.

In this paper, we present the design and implementation of *Topnet*, an extension of the top UNIX utility that introduces a *process-centric* approach to network traffic monitoring. *Topnet* provides users with a simple and intuitive real-time attribution of the system's incoming and outgoing network traffic to the running processes in additional columns of the familiar console output of top. By treating incoming and outgoing traffic as additional process properties, the user can instantly spot the running processes with current network activity, and by sorting according to the relevant column, identify which applications send or receive most of the traffic.

Our work suggests that *Topnet* through (i) a familiar top user interface and (ii) a reasonable performance overhead, provides an accurate way to attribute network traffic to individual processes, enabling users to have a better understanding of the consumption of resources in their systems.

The contributions of this paper are:

- We provide a *process-centric approach* to processor, memory, and network resource monitoring, that is implemented completely in user-level without requiring any kernel modifications as previous approaches do.

[1]The authors are also with the University of Crete.

- We *demonstrate the feasibility* of our approach by providing an open source implementation of *Topnet*.
- We *evaluate Topnet* and show that it provides high measurement accuracy at a reasonable computational cost for most of the expected operation range.

Related Work

In this section we present related efforts. We start with exploring modern home networks and the arising need for better administration, even by the end user and then we proceed in more complex tasks like monitoring and classifying aggregated network traffic. We finally, present visualization techniques for similar applications to *Topnet*.

Home Networks

The evolution of home networks and the complexity they inhabit lately has driven the community to seek algorithms and technologies for a better administration of a host by the end user. Broadband technologies and their characteristics have been studied by Dischinger, et al. [19]. According to this study, broadband technologies that appear often in home networks experience high jitter rates, affecting the expected performance.

Papagiannaki, et al. [42], has shown how small configuration changes can affect the network performance of hosts connected in a wireless media of a home environment. In this context, it is vital for a user to have a tool that will assist her in inspecting the network health of a running host as per process network utilization is concerned. Indeed, this need is profound; lately there have been proposed standards [4] for per application network statistics to be included in modern operating systems. This effort will further assist the development of applications that collect and present per process network accounting information.

Network Monitoring Systems

Over the past few years, there has been an increasing interest in passive network monitoring systems and tools. Indeed, several infrastructures are available for deployment by system and network administrators in order to monitor the traffic usage of their networks by parsing either raw network packets, SNMP data, or network flow data.

For example, MRTG [41] is a popular tool for monitoring SNMP network devices and visualizing network usage. FlowScan [43] and commercial systems including Cisco Network Analysis Module Software [12] and IBM Aurora [25] present traffic usage patterns using Netflow exported data, and classify traffic using mainly the IANA port number list.[2] AutoFocus [20] aims at identifying important network consumption by clustering flows of similar or same interest, i.e., a large number of small network flows by a single web server that will not be positioned at the top flows if considered one by one.

Although these systems are widely deployed, they aim to provide a ''bird's eye view'' of the network traffic, informing system administrators of the overall status of their network and alerting them of major traffic events. On the contrary, *Topnet* focuses on the traffic usage not of entire networks, but of *individual applications* running on specified end host computers, and thus present a functionality complementary to these systems.

We consider, though, atop [1] to be the tool most like *Topnet* in terms of operation. atop provides a complete set of process resource statistics (RAM, CPU, storage, network, etc.). However, *Topnet* differentiates from atop in two basic ways. First, atop needs kernel modifications and thus has deployment complications, whereas *Topnet* uses widely used libraries for packet capturing in user-space, and second, *Topnet* is built over a well known and trusted framework that is well established in the community: the classic UNIX top tool.

Network Packet Capturing Systems

To provide a better understanding of network traffic, some systems enable administrators to capture (and subsequently process) all traffic which passes through a router (or computer). The popular libpcap [37] packet monitoring library provides a portable Application Programming Interface (API) for user-level packet capture. The libpcap interface supports a filtering mechanism based on the BSD Packet Filter [36], which allows for selective packet capture based on packet header fields. The Linux Socket Filter [27] offers similar functionality with BPF, while xPF [28] and FFPF [10] provide a richer programming environment for network monitoring at the packet filter level. Iftop [47] is a libpcap-based application that reports the bandwidth usage of a network interface by displaying a breakdown of the network traffic according to the active network flows.

To improve the functionality of monitoring sensors, beyond the naive capturing of network packets, FLAME [7] allows users to directly install custom modules on the monitoring system, similarly in principle to Management-by-Delegation models [23]. Windmill [35] is an extensible network probe environment which allows loading of ''experiments'' on the probe for analyzing protocol performance.

To provide intelligent packet capturing and sophisticated packet processing in a single system, in our earlier work, we designed and implemented MAPI [44, 46] an Application Programming Interface for network monitoring. MAPI captures packets passing through a monitoring link and provides mechanisms to build custom applications for network monitoring. Hughes in [24] presented qcap: a library for capturing and decoding live traffic streams in network level. The

[2]http://www.iana.org/assignments/port-numbers

library provides the functionality for developing network monitoring applications able to decompose the flows up to the layer 7 protocol.

Although packet-capturing systems may provide a global view of network traffic and of the IP addresses which generate/consume the observed traffic, they are usually fined-tuned for deployment at network nodes and not at end hosts. Thus, these systems provide little, if any, information on the end user *applications* which generate the traffic in question. In contrast, our approach, *Topnet*, targeted for deployment at end hosts, aims to help end users as well as system administrators understand the individual applications which are responsible for the network traffic generated from (or destined to) a particular computer.

Traffic Classification Tools

To improve the understanding of the types of applications which generate a particular amount of traffic in the network, several packet- or flow-capturing systems are equipped with traffic-classification tools. These tools are able to attribute network traffic to classes of applications that generated this traffic, such as peer-to-peer systems, web browsing, and IP telephony.

The first generation of traffic classification tools attributed traffic to applications based on the IANA port number list, associating, for example, all traffic destined to (or originated from) port 80 with web browsing, all traffic destined to port 25 with email, and so on [43]. However, in the wake of elusive peer-to-peer applications that use dynamic ports, the accuracy of this port-based approach was quickly shown to be inaccurate [39, 30]. As a result, recent approaches use deep packet inspection and application signatures for attributing traffic flows to the corresponding applications [29, 45].

NetADHICT [26] provides a hierarchical decomposition of traffic, based on similar patterns both in header and payload level, yet the labeling of the nodes derived from the application is left for the administrator. Another application that can classify network traffic by packet inspection is ntop [17, 18]: an extended network monitoring tool for displaying the top users of a network. It uses a plugin architecture for deploying decoders that will assign the network flows to the applications that generated them. Finally, Appmon [8] is an open-source passive network monitoring application that applies deep packet inspection and searches the network packets for specific application signatures.

Although highly accurate, methods based on deep packet inspection suffer from two major drawbacks: (i) they have high computational costs, and (ii) they are ineffective in the presence of encryption. To overcome these deficiencies, recent approaches try to identify the applications that generate the traffic by not looking at the packet payload, but only at the transport layer [31, 13] or at the statistical characteristics of the network flow, like packet sizes and round-trip times [9, 50, 15, 14]. A different approach presented by Karagianis, et al. tries to classify traffic by characterizing the behavior of the host generating this traffic [32].

Although the previously described traffic-classification tools have been widely used, they have been developed for deployment at network nodes and may not appropriate for host-level use. Although the motivated user could run such a tool, e.g., Appmon [8], for monitoring a single host, it will still suffer from several drawbacks:

- Appmon will not be able to categorize encrypted traffic.
- Appmon has no notion of user-level applications. That is, Appmon reports how much ftp traffic is generated by an IP address, but does not know which process generated the traffic.
- Appmon can not distinguish between similar applications. For example, if the ftp traffic is generated by two or more ftp clients, Appmon will not be able to report the ftp traffic generated by each one of the clients.

On the contrary, *Topnet* provides all the above functionality with the same "look and feel" of the familiar top UNIX application.

Visualization

Work has also been done in visualization of network data with the goal to provide the administrator with an easily understandable picture of the network. VisFlowConnect [49], NVisionIP [33] and PortVis [38] are all tools providing connection level visualization of the network hosts in order to identify suspicious network usage though they lack host level information that would be able to identify the exact process responsible for the suspicious traffic. *Topnet* can tie up loose ends by providing the user of these tools with the real source of the traffic and enable her to decide for the proper administration actions to be taken.

Personal firewall tools like ZoneAlarm [11] and Objective Development's Little Snitch [40] provide the user with sufficient information about the network usage experienced by running applications. But these tools are limited to a single operating system, and explicitly target the user of the machine and are of little to zero use for the administrator.

Fink, et al. implemented HoNe, a host-network visualization tool for packet-process correlation [22, 21]. Their tool is based on a loadable kernel module and the netfilter packet filtering framework [5]. HoNe focuses on visualizing the number of network connections a machine has and the corresponding applications for that connections. It mainly targets security analysis, through visualizing the process of an intrusion attempt, malware download and operation, in a connection level. Although HoNe shares some aspects with *Topnet*, our approach has several advantages including:

- HoNe needs kernel modifications while *Topnet* is completely implemented in user level.
- *Topnet* provides the simple and familiar "look and feel" of the *top* UNIX utility.
- Although HoNe does not provide any performance analysis, based on its description we believe that *Topnet* is faster and scales to larger network speeds more easily.

To summarize, we believe that *Topnet* fills a clearly identified gap in host-level network traffic monitoring. *Topnet* capitalizes on popular network level traffic monitoring (libpcap) and a proven resource monitoring system (*top*). *Topnet* provides an easy-to-use tool which helps users and administrators understand the traffic generated by the applications running on their computers.

Application Description

Implementation

Our implementation goal was to provide a tool that will be as efficient as possible, provide the maximum functionality and would be easily installed on any system. Thus we decided to build our tool based on existing, widely used system tools. We used the top UNIX utility as an interface, since it is widely used for system monitoring and easily understandable. More precisely, we used the Linux version of top. The choice of Linux was made for convenience, since we had easy access to a Linux based testbed. As we point out in the Future Work Section, we plan to port *Topnet* to various flavors of UNIX.

Although top provides comprehensive monitoring of the computing and memory resources consumed by the running processes, it lacks information about the network usage of the processes. To provide this functionality, we needed (i) a way to read the incoming and outgoing packets and (ii) a way to correlate the network traffic flows with the process responsible for them.

To read the packets from the network and correlate them with the process responsible for sending/receiving them, we used the widely available libpcap and netstat tools. Both tools are available for both UNIX-like and Windows systems and are widely deployed and used. Using libpcap, we implemented a simple sniffer that reads packets from the network interface. The sniffer is implemented as part of top's source code. At initialization, it tries to find the default network interface of the machine and next starts reading all the packets coming to and going from that interface. For each network flow, our application keeps an entry in a hash table where the correlation and statistics take place.

When a new packet is received by libpcap, *Topnet* first looks if the flow it belongs to has already been attributed to a process. If so, the statistics of the flow are updated and the application moves on to the next packet. If the flow has not been assigned to a process yet, we make a call to the netstat function to correlate the newly observed flow with the corresponding process. netstat reads various network-related structures and outputs results about active sockets, network protocol traffic statistics, remote endpoints and routing information. We use part of the netstat functionality to retrieve the list of active network sockets on the system. Having that information, we can assign the newly observed network flow to the process responsible for it.

Fink, et al. [21] argue that netstat would not be a complete solution since it will fail to track short-lived flows due to the fact that it polls the kernel with a user specified rate. Due to polling, a short-lived flow may not be captured by netstat in case it is created and destroyed between two consecutive polling periods. In contrast to netstat, we address with this limitation by reading the relevant data from the kernel whenever *Topnet* encounters a new packet belonging to a non-correlated flow. This allows us to track the new flow as soon as its first packet arrives, before the flow is terminated, even if the flow is short-lived.

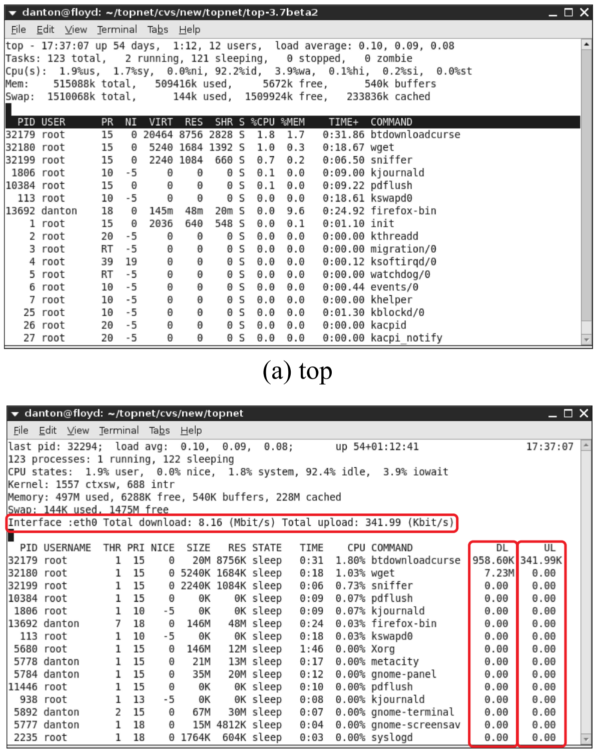
Periodically, every five minutes, the application performs a full pass of the hash table in order to clean invalid entries. We consider a flow entry to be invalid if it neither received nor transmitted any data for a period of 60 seconds. This way, the hash table contains only the active flows of the machine and minimizes the memory overhead of the application.

Features

The *Topnet* default window (see Figure 1b) is almost identical to the window of the top utility (see Figure 1a), with the addition of two more columns next to the existing columns. The first column shows the incoming traffic rate for each process, while the second shows the outgoing traffic rate. The default values are in Kbps per second. We should note here, that *Topnet* reports traffic from a process perspective. That is, it counts only the payload bytes (layer 4) of each packets. Though, since it does not reassemble flows, bytes belonging to retransmissions are also going to be reported.

Also in the general statistics header, the upper lines in the *Topnet* interface, we added a line that reports the traffic speed of the default interface of the monitored machine. Figure 1b shows the default interface of *Topnet*. As we can see *Topnet* informs its user for the rate of the network consuming process. In this figure we have a wget process downloading a file, and a BitTorrent client with an active BitTorrent download.

Figure 2 presents the *Topnet* window in more detail. Here we show only the processes that use the network. In the top two rows of the per-process utilization part we see the three applications that make use of the network during the time the screenshot was taken. These three processes are operating three file transfers, an HTTP download using wget, an active



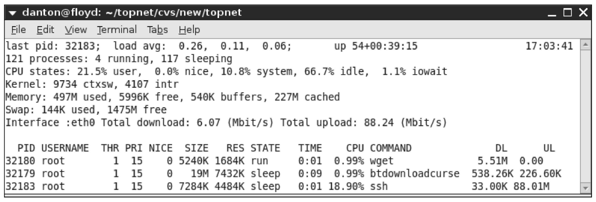
```
danton@floyd: ~/topnet/cvs/new/topnet/top-3.7beta2
top - 17:37:07 up 54 days,  1:12, 12 users,  load average: 0.10, 0.09, 0.08
Tasks: 123 total,   2 running, 121 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.9%us,  1.7%sy,  0.0%ni, 92.2%id,  3.9%wa,  0.1%hi,  0.2%si,  0.0%st
Mem:    515088k total,   509416k used,     5672k free,      540k buffers
Swap:  1510068k total,      144k used,  1509924k free,   233836k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
32179 root      15   0 20464 8756 2828 S  1.8  1.7   0:31.86 btdownloadcurse
32180 root      15   0  5240 1684 1392 S  1.0  0.3   0:18.67 wget
32199 root      15   0  2240 1084  660 S  0.7  0.2   0:06.50 sniffer
 1806 root      10  -5     0    0    0 S  0.1  0.0   0:09.00 kjournald
10384 root      15   0     0    0    0 S  0.1  0.0   0:09.22 pdflush
  113 root      10  -5     0    0    0 S  0.0  0.0   0:18.61 kswapd0
13692 danton    18   0  145m  48m  20m S  0.0  9.6   0:24.92 firefox-bin
    1 root      15   0  2036  640  548 S  0.0  0.1   0:01.10 init
    2 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      39  19     0    0    0 S  0.0  0.0   0:00.12 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      10  -5     0    0    0 S  0.0  0.0   0:00.44 events/0
    7 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
   25 root      10  -5     0    0    0 S  0.0  0.0   0:01.30 kblockd/0
   26 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   27 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify
```

(a) top

```
danton@floyd: ~/topnet/cvs/new/topnet
last pid: 32294;  load avg:  0.10,  0.09,  0.08;       up 54+01:12:41            17:37:07
123 processes: 1 running, 122 sleeping
CPU states:  1.9% user,  0.0% nice,  1.8% system, 92.4% idle,  3.9% iowait
Kernel: 1557 ctxsw, 688 intr
Memory: 497M used, 6288K free, 540K buffers, 228M cached
Swap: 144K used, 1475M free
Interface :eth0 Total download: 8.16 (Mbit/s) Total upload: 341.99 (Kbit/s)

  PID USERNAME  THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND              DL      UL
32179 root        1  15    0   20M 8756K sleep   0:31  1.80% btdownloadcurse 958.60K 341.99K
32180 root        1  15    0 5240K 1684K sleep   0:18  1.03% wget              7.23M   0.00
32199 root        1  15    0 2240K 1084K sleep   0:06  0.73% sniffer            0.00   0.00
10384 root        1  15    0    0K    0K sleep   0:09  0.07% pdflush            0.00   0.00
 1806 root        1  10   -5    0K    0K sleep   0:09  0.07% kjournald          0.00   0.00
13692 danton      7  18    0  146M   48M sleep   0:24  0.03% firefox-bin        0.00   0.00
  113 root        1  10   -5    0K    0K sleep   0:18  0.03% kswapd0            0.00   0.00
 5680 root        1  15    0  146M   12M sleep   1:46  0.00% Xorg               0.00   0.00
 5778 danton      1  15    0   21M   13M sleep   0:17  0.00% metacity           0.00   0.00
 5784 danton      1  15    0   35M   20M sleep   0:12  0.00% gnome-panel        0.00   0.00
11446 root        1  15    0    0K    0K sleep   0:10  0.00% pdflush            0.00   0.00
  938 root        1  13   -5    0K    0K sleep   0:08  0.00% kjournald          0.00   0.00
 5892 danton      2  15    0   67M   30M sleep   0:07  0.00% gnome-terminal     0.00   0.00
 5777 danton      1  18    0   15M 4812K sleep   0:04  0.00% gnome-screensav    0.00   0.00
 2235 root        1  18    0 1764K  604K sleep   0:03  0.00% syslogd            0.00   0.00
```

(b) topnet

Figure 1: Screenshots of (a) the Linux top utility and (b) the *Topnet* tool.



```
danton@floyd: ~/topnet/cvs/new/topnet
last pid: 32183;  load avg:  0.26,  0.11,  0.06;       up 54+00:39:15            17:03:41
121 processes: 4 running, 117 sleeping
CPU states: 21.5% user,  0.0% nice, 10.8% system, 66.7% idle,  1.1% iowait
Kernel: 9734 ctxsw, 4107 intr
Memory: 497M used, 5996K free, 540K buffers, 227M cached
Swap: 144K used, 1475M free
Interface :eth0 Total download: 6.07 (Mbit/s) Total upload: 88.24 (Mbit/s)

  PID USERNAME  THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND              DL      UL
32180 root        1  15    0 5240K 1684K run     0:01  0.99% wget              5.51M  0.00
32179 root        1  15    0   19M 7432K sleep   0:09  0.99% btdownloadcurse 538.26K 226.60K
32183 root        1  15    0 7284K 4484K sleep   0:01 18.90% ssh              33.00K 88.01M
```

Figure 2: The *Topnet* window showing the network usage of three applications.

BitTorrent download, and an ssh file copy from another machine.

To ease the use of *Topnet* and to make as much information as possible available instantly, we added some hot keys to *Topnet*'s user interface. Table 1 presents the keys we added and the effect they have on the data presentation.

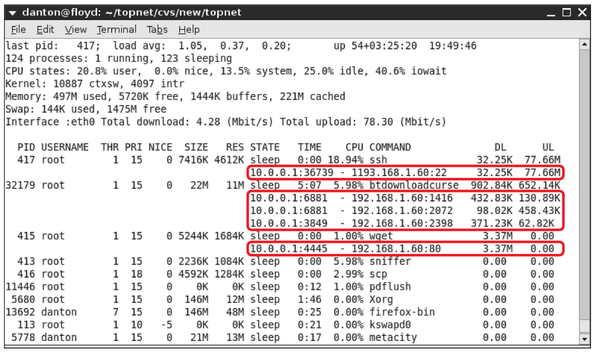
Hot keys can be used to highlight the applications with the most network activity. Pressing *r*, *t* and *R* sort processes according to the inbound, outbound or total traffic respectively. Using *a*, the user is able to select information about just the processes which are responsible for the network load of the machine.

Pressing *F* toggles the presentation of active network flows per process. The displayed information includes a list with the process' active connections and the network utilization for each connection. As an example, Figure 3 presents the full information available by *Topnet*'s interface. For each process *Topnet* displays the incoming, outgoing and total traffic rate per update interval in the process line, as in the default case of running *Topnet*.

For each process with active network connections, *Topnet* displays these connections as well as the traffic rate of each connection. In Figure 3 we see the analysis for three network consuming process running in a machine. Two of them (wget and ssh) have one flow each, while the third one, a BitTorrent process, has three active flows sharing its bandwidth.

Key	Functionality
r	Sort by download rate (receiving)
t	Sort by upload rate (transmitting)
R	Sort by total Rate
F	Toggle the display of network Flows per process
a	Show only processes with network load (active)
D	Dump processes traffic to tcpdump file format
y	Toggle the display of unknown network flows
$	Monitor a different interface

Table 1: Function keys and their functionality.



```
danton@floyd: ~/topnet/cvs/new/topnet
last pid:   417;  load avg:  1.05,  0.37,  0.20;       up 54+03:25:20  19:49:46
124 processes: 1 running, 123 sleeping
CPU states: 20.8% user,  0.0% nice, 13.5% system, 25.0% idle, 40.6% iowait
Kernel: 10887 ctxsw, 4097 intr
Memory: 497M used, 5720K free, 1444K buffers, 221M cached
Swap: 144K used, 1475M free
Interface :eth0 Total download: 4.28 (Mbit/s) Total upload: 78.30 (Mbit/s)

  PID USERNAME  THR PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND              DL      UL
  417 root        1  15    0 7416K 4612K sleep   0:00 18.94% ssh              32.25K  77.66M
                                       10.0.0.1:36739 - 1193.168.1.60:22      32.25K  77.66M
32179 root        1  15    0   22M   11M sleep   5:07  5.98% btdownloadcurse 902.84K 652.14K
                                       10.0.0.1:6881  - 192.168.1.60:1416    432.83K 130.89K
                                       10.0.0.1:6881  - 192.168.1.60:2072     98.02K 458.43K
                                       10.0.0.1:3849  - 192.168.1.60:2398    371.23K 62.82K
  415 root        1  15    0 5244K 1684K sleep   0:00  1.00% wget              3.37M   0.00
                                       10.0.0.1:4445  - 192.168.1.60:80        3.37M   0.00
  413 root        1  15    0 2236K 1084K sleep   0:00  5.98% sniffer            0.00   0.00
  416 root        1  18    0 4592K 1284K sleep   0:00  2.99% scp                0.00   0.00
11446 root        1  15    0    0K    0K sleep   0:12  1.00% pdflush            0.00   0.00
 5680 root        1  15    0  146M   12M sleep   1:46  0.00% Xorg               0.00   0.00
13692 danton      7  15    0  146M   48M sleep   0:25  0.00% firefox-bin        0.00   0.00
  113 root        1  10   -5    0K    0K sleep   0:21  0.00% kswapd0            0.00   0.00
 5778 danton      1  15    0   21M   13M sleep   0:17  0.00% metacity           0.00   0.00
```

Figure 3: *Topnet* showing the list of active connections for three processes with network activity.

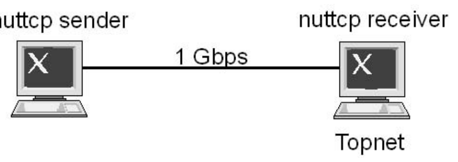


Figure 4: The testbed used for the evaluation of *Topnet*.

One of the most useful functions of *Topnet* is the ability to log the traffic of a specific application to a dump file for later use and analysis. This is done with the *D* key followed by the specific process id of interest. The raw packets are dumped in a file in the directory from which top was started with the PID and the process name in the filename.

Finally, using *y* one can have a view of the network flows not known to belong to a specific process. This case may be very useful when a user hides from *Topnet*, for instance, by using an application that uses raw sockets for communication. The *Topnet* utility, since it is based in libpcap, can capture packets sent using raw sockets. However, it is hard to map this traffic to a specific process. In order to cope with this issue, we group all *orphan traffic* – all traffic that we can not assign to a process – in a special *Topnet* window.

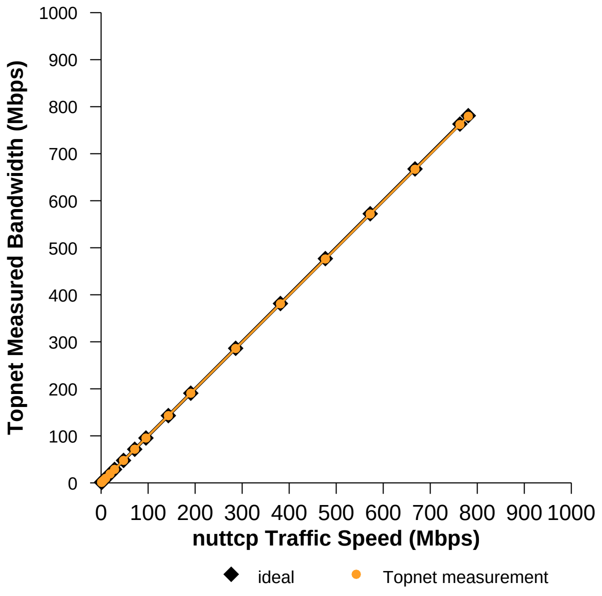


Figure 5: *Topnet* reported network consumption for the artificial traffic created with nuttcp.

Evaluation

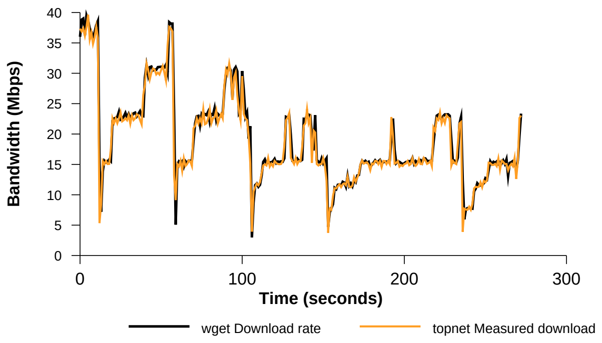
In this section we provide a twofold evaluation of our application. First we show the accuracy of the application using (i) artificial traffic in a controlled testbed environment, and (ii) real world traffic from two popular application protocols, HTTP and BitTorrent. Next, we present the resource overhead consumed by *Topnet*.

Experimental Environment

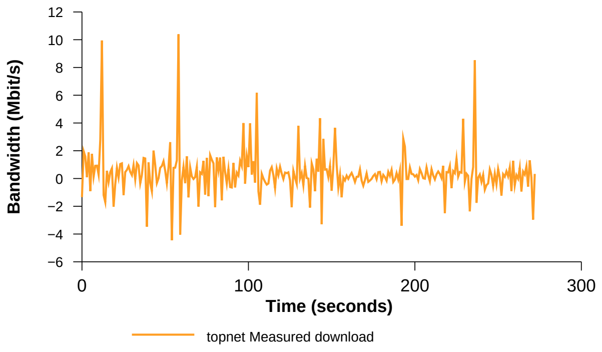
For all the experiments presented in this section we used a Dell 1420 computer equipped with a Dual Xeon 2.8 MHz CPU and 512 Mbyte of memory. The machine was connected to the network using a commodity Gigabit network interface.

Accuracy

The first and most important test for the application was to evaluate its accuracy in measuring the network load of a specific process. We first used artificial traffic in a testbed environment and as a next step we evaluated the accuracy using real world HTTP and BitTorrent traffic. In all our experiments we experienced zero packet loss with tcpdump, therefore those measurements represent ground truth for comparison.



(a) wget vs. topnet



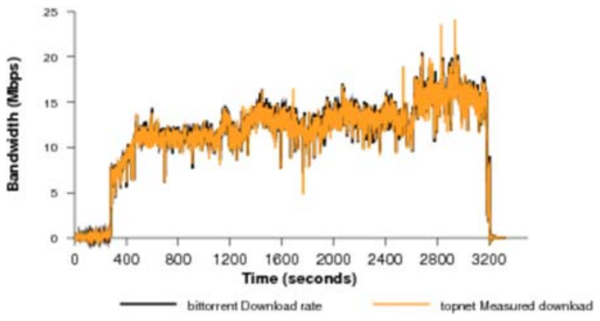
(b) Inaccuracy

Figure 6: *Topnet* results while monitoring a web download compared with the traffic reported by a tcpdump running in parallel. Figure (a) shows the actual traffic seen by both tcpdump and *Topnet* during the lifetime of the experiment. Figure (b) shows the difference between the two measurements. With an average accuracy of 98.1%, in most cases *Topnet* provides accurate results with the exception of some short reporting periods.

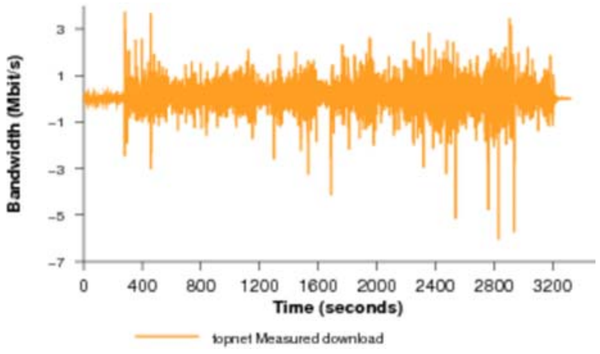
Measuring Synthetically-Generated Traffic

For the testbed experiments we used two machines on a Gigabit network, exchanging traffic with each other using nuttcp [3]. One machine was used as the sender and the other as receiver. We ran *Topnet* on the receiver machine and logged the measurement results per second for the nuttcp receiver process. We instrumented the nuttcp sender to send traffic to the receiver in different rates, ranging from 1 Mbps to about 870 Mbps. The complete testbed is depicted in Figure 4.

Figure 5 presents the measured traffic load reported by *Topnet* during the experiment. The *x* axis presents the traffic send through nuttcp in Mbps, while the *y* axis shows the traffic measured by *Topnet*. The results are averages for the total of 10 runs for each traffic speed and present the average traffic sent and measured for the duration of each run. Figure 5 also plots the $y = x$ *ideal* curve. We see that *Topnet* has an almost exact match with the ideal in all ranges of the experiments. Our data suggest that there is less than 0.1% difference between the two curves.



(a) torrent vs. *Topnet*



(b) Inaccuracy

Figure 7: *Topnet* results while monitoring a BitTorrent client compared with the traffic logged by tcpdump. Figure (a) shows the actual inbound traffic measured by the two tool, *Topnet* manages to follow the exact traffic rates of the BitTorrent download. Figure (b) shows the differences from the two measurements in more detail. Again, *Topnet* seems to show only slight differences from the tcpdump traffic, with the overall accuracy to be 98.8%.

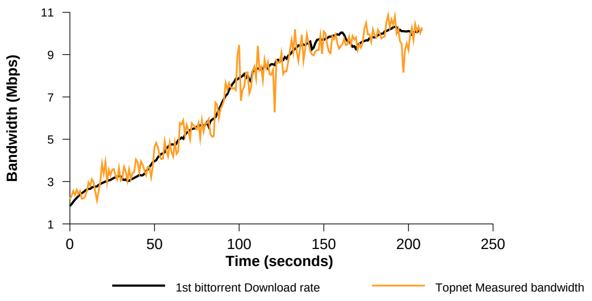
Measuring HTTP and BitTorrent Clients

Since *Topnet* shows high accuracy on a controlled environment with artificial traffic, the next step was to evaluate the measurement accuracy using real world traffic. We used two applications utilizing two popular protocols: HTTP and BitTorrent.

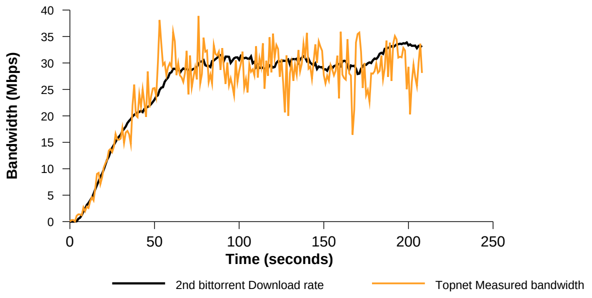
For the evaluation with HTTP traffic we used the well known wget application, a command-line web HTTP GET application. We used tcpdump to log the HTTP traffic exchanged by the machine during the time of the experiment and compared it with the traffic reported by *Topnet*. *Topnet* was instrumented to report the per-application traffic every second.

We used wget to download a Linux distribution ISO file.[3] Figure 6a presents the incoming traffic reported by *Topnet* during the HTTP download in comparison with the traffic reported by tcpdump. The relative time for the whole duration of the download is plotted on *x* axis, while the measured one-way traffic (inbound) in Mbps is plotted on the *y* axis. Each point

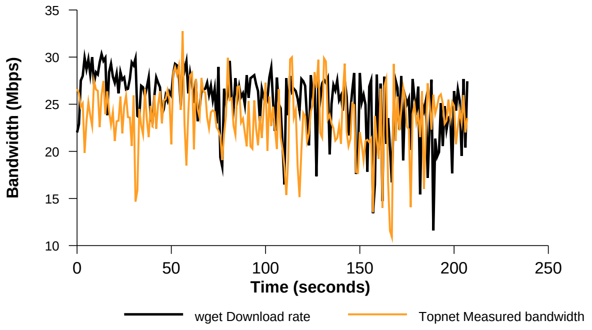
[3]http://ftp.belnet.be/mirror/ubuntu.com/releases/hardy/ubuntu-8.04.1-desktop-i386.iso .



(a) BitTorrent 1



(b) Bittorrent 2



(c) Wget

Figure 8: *Topnet* reported traffic rates when monitoring one wget and two BitTorrent downloads in parallel. Both measurements for BitTorrent show an average difference around 1% (1.05% for client 1 and 0.43% for client 2). The wget measurement shows a larger difference around 10%.

gives the traffic for one second. The figure shows that *Topnet* managed to accurately measure the traffic of the download during its whole duration and also follows the changes experienced during the download. Figure 6b plots the difference in the measurement reported by each tool, as the inaccuracy of our tool. As we see, *Topnet* experiences limited differences from the tcpdump logged traffic. This differences can be explained from slightly different times for reporting the traffic.

For the next experiment we used the BitTorrent client to download a Linux distribution through the BitTorrent protocol.[4] We once again used tcpdump to

[4]http://cdimage.debian.org/debian-cd/4.0_r3/i386/bt-dvd/debian-40r3-i386-DVD-1.iso.torrent

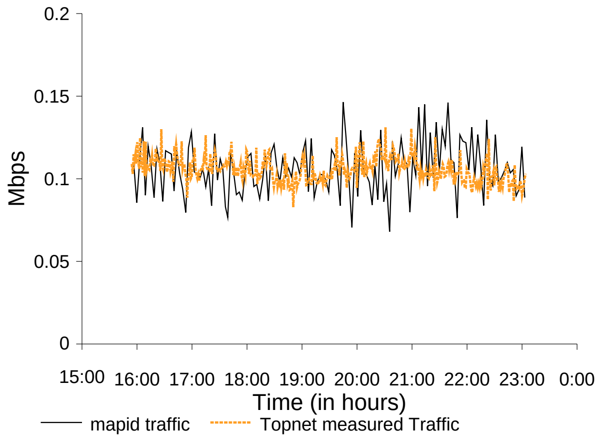


Figure 9: *Topnet* accuracy while monitoring a server process for several hours. The overall difference between *Topnet* and tcpdump is 1.73%.

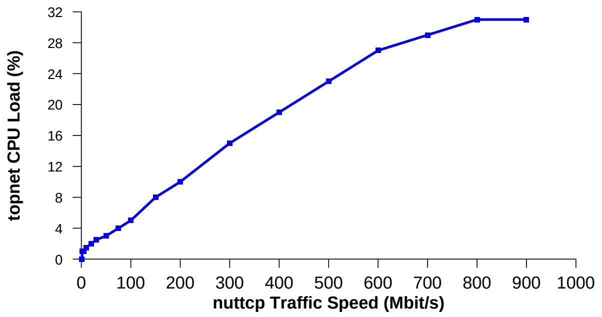


Figure 10: *Topnet* CPU usage using artificial traffic created with nuttcp. *Topnet* uses at most 31% of the CPU while experiencing a full network loaded machine with 900 Mbps of traffic.

report the downloaded traffic per second for the whole duration of the data transfer.[5] Figure 7a reports the true traffic exchanged by a BitTorrent client while downloading the file, as incoming traffic in bits per second, in comparison with the traffic reported by *Topnet* for the downloading process. We see that the results reported by *Topnet* and the BitTorrent client are very close. To better show the accuracy of *Topnet* we plot the difference between the two applications for each measurement second in Figure 7b. *Topnet* measurements exhibit little differences from tcpdump-based measurements, with the exception of a small number of outliers. The differences can be explained with the different timing that the two applications have, due to manual start-up.

For the next experiment, we tried to simulate the everyday network usage of a normal user. For instance, the user might browse someweb pages while there may be some active BitTorrent downloads in the background. To simulate this case we concurrently employed an HTTP download and two BitTorrent downloads for different files. The comparison is pictured in Figure 8.

[5]During this experiment, access was limited only to the BitTorrent client so as to avoid unexpected non-BitTorrent flows.

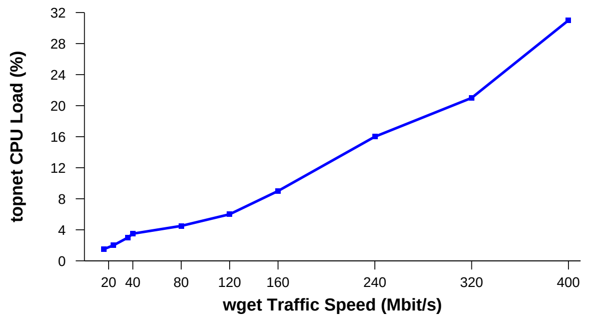


Figure 11: *Topnet* CPU load for several HTTP downloads, each at a different network speed.

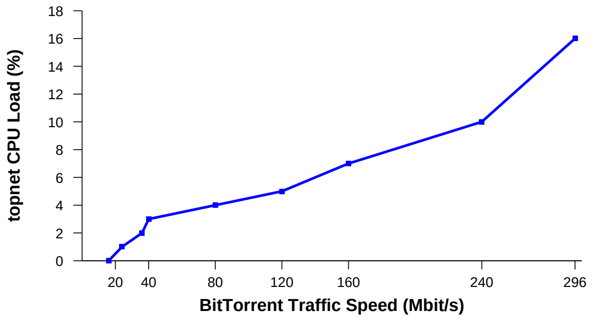


Figure 12: *Topnet* CPU load for several BitTorrent downloads, each at a different network speed. *Topnet* uses less than 10% of the CPU while experiencing download rates at 240 Mbps.

Topnet manages to report the traffic with high accuracy (differences are around 1%). Also in the case of the wget download, where the traffic rate experiences large changes between the measurement periods, *Topnet* follows these changes.

Comparison to Other Traffic Measurement Tools

To further evaluate the measurement accuracy of *Topnet* we installed *Topnet* on a server and monitored outgoing traffic for several hours. The server machine was running mapid, a distributed monitoring daemon, which allows users to run remote monitoring applications. That is, it provides the monitoring capabilities and allows the user to get the information in which she is interested. As an example, the user may ask from the monitoring daemon to count all packets coming in and out from a specific subnet, and periodically will ask for the results. We used tcpdump to log all the traffic coming in and out of the machine using mapid's port. At the same time we were running *Topnet* on the machine measuring the traffic for the specific process number. The comparison of the two traffic measurements is shown in Figure 9. *Topnet*, manages to follow the traffic consumption experienced by the server process. The figure shows one minute averages of the traffic during a period of seven hours.

Performance Measurements

The second part of our evaluation identifies and quantifies the performance overhead that *Topnet* induces during the monitoring process.

The first performance measurement was done in the testbed during the nuttcp experiment presented in the previous subsection. We used the systems' top utility to measure the CPU overhead induced by *Topnet* during the different nuttcp traffic exchanges. The results are presented in Figure 10. The *x* axis plots the exchanged traffic rate in Mbps and the *y* axis shows the average CPU percentage used by *Topnet* during each measurement as a function of the traffic measured. We see that the CPU overhead of *Topnet* is almost proportional to the traffic measured. This should be expected as the main overhead of *Topnet* is attributed to the libpcap library, which is being used to read the packets from the network card. We observe, however, that for low network traffic, i.e., under 200 Mbps, the CPU usage is below 10%.

To evaluate the performance of *Topnet* using real world traffic, we once again used HTTP and BitTorrent traffic. As in earlier experiment, we used wget to download the Linux ISO image mentioned earlier. To understand the impact of network traffic on the performance overhead of *Topnet*, we rate-limited wget to download speeds ranging from 20 Mbps to 400 Mbps. Figure 11 shows the CPU load used by *Topnet* for the different download rates used by wget. As seen previously, the performance of *Topnet* is proportional to the network traffic monitored, staying below 10% as long as the traffic received is less than 200 Mbps.

We repeated the previous experiment using the BitTorrent client, instead of wget. The results, shown in Figure 12, follow the same pattern: i.e., the performance overhead of *Topnet* is proportional to network traffic observed.

Our experiments with a commodity PC suggest that as long as the network traffic observed is relatively low, i.e., up to a few tens Mbps, the performance overhead of *Topnet* is usually below 5%, which we consider acceptable for occasional use, given the information it provides the administrator. We understand though that under very heavy traffic, i.e., several hundreds of Mbps, the performance penalty of *Topnet* may become significant and the tool might experience packet losses. Current trends, however, suggest that this performance overhead, which is completely attributed to the overhead of the packet capture library, tends to decrease with time, as more optimized versions of libpcap are being implemented and deployed (PFring [16], pcap-mmap [48]). Using the aforementioned versions of libpcap, the overhead from copying packets from system to user level decreases, and any packet loss disappears. In our current case, since no packet loss was experienced, running *Topnet* compiled with pcap-mmap gave the same performance measurements.

Memory Usage

As mentioned earlier, *Topnet* uses a hash table to hold all active flows on the machine. During our experiments, where the number of active flows was small, *Topnet* experienced memory overhead below 1%, in all cases.

Reporting Period

In all of our experiments we used 1 second as a reporting period from *Topnet*. Changing the reporting period to a larger number of seconds does not affect the performance overhead of the application, since the CPU utilization is due to libpcap, for reading packets from the network and transferring them to the application. Though, increasing the reporting period may smooth the reported network measurement and reduce any discrepancies observed.

Use Cases

In this section we discuss various real world scenarios in which *Topnet*'s usage could be beneficial. This discussion unveils the strength of *Topnet* in every day situations and incidents. We explore the use of *Topnet* by three target groups: administrators, researchers and end users. In every use case, we focus on the convenience provided to the end user by *Topnet*.

Administrators

Observing system performance is vital to the system administrator. The classic UNIX top utility is designed towards that direction, giving summaries for CPU and memory consumption per process. Our extended top, *Topnet*, broadens this model by adding inspection for network resources consumption per process. Understanding the network usage of an individual process has become increasingly important, if not more important than, as understanding CPU and memory resource consumption.

A summary of network consumption per host, something that most operating systems provide by default, is not sufficient. There is a need for greater *per-process* granularity. This is especially important for tracking security threats like rootkits, trojan horses, and other malware, which consume network resources in a stealthy way, without altering the overall condition of the host as far as the network consumption is concerned. More precisely, a malicious program that performs IP/port scanning or communicates with other members of a Botnet, may never be spotted by just observing the total outbound traffic of a host, since the traffic exported by the malicious program has a low volume. However, this malicious program will have active flow records in the *Topnet* utility and thus it will be easily spotted by an administrator.

Researchers

System research, very often, involves the inspection and measurement of network capabilities of an application. Sometimes, the researcher has to correlate consumption in all available resources, namely CPU, memory and network. The most well known tactic is to use multiple tools for the inspection of different kind of resources. For example, the top UNIX utility may be used for the inspection of CPU/memory resources and an application trace collected with tcpdump, or another packet capturing tool, can be used to inspect the network utilization experienced by the application.

Topnet combines all above tasks to one program able to monitor in real-time an application's utilization for all kind of resources. Thus, *Topnet* offers a convenient way for the researcher to inspect in real-time the behavior of an application and gives immediately an indication of its resource consumption.

End Users

End users' systems have drastically evolved and become more sophisticated as far as the amount of running applications and their complexity is concerned. In addition, end users are frequently mobile, utilizing wireless access media. As an example, excessive bandwidth consumption might significantly reduce the lifetime of the battery of a user's laptop. A tool such as *Topnet* that can inform the user of such events is considerably valuable.

We believe that the increased complexity of a modern home machine (a desktop PC, or a laptop) has driven the home user to cope with tasks that are usually performed by an administrator. The user, in a sense, frequently becomes *an administrator of her own system*. Thus, we propose *Topnet* as a convenient tool for the quick inspection of the resource utilization of a user's machine. With increasing popularity of networking applications (Web surfing, IM, file-sharing, etc.), inspecting the network activity per process significantly changes the picture the user has about the actual condition of her machine.

Communication Patterns

The cases above depict various classes of users that receive benefit and convenience from the *Topnet* tool. In contrast with traditional programs that experience network utilization upon user activity, modern applications remain active even when the user is not explicitly using them. This *background behavior* causes significant network activity, without easily being spotted by the user.

In Table 2 we list a few representative applications and their background behaviors, i.e., network activity not explicitly initiated by the user. In all these application profiles, *Topnet* can easily expose such background network activity.

Future Work

Topnet fills a substational gap to current system administration by monitoring network demands at the process level. However, we believe that *Topnet* can be further enhanced. In this Section we list some features we plan to implement in following versions of the tool.

Security Enhancements

As we have pointed out in Use Cases Section, *Topnet* can be used for tracking security threats, like rootkits, trojan horses and other malware. In such a case, the administrator can inspect network consumption, related to malicious activity rather than a legitimate user application. We, consider this feature important to expose security threats. Thus, we plan to implement an accounting feature in *Topnet* in order to give system administrators the opportunity to get a better global view of the monitored network.

We plan to have a *Topnet* mode that logs all network activity per process to a file. Then, it would be easy to create a script that collects all log files from *Topnet*-enabled hosts running in the network and combine the information to one file. This file can then be processed for malicious activity. For example, someone can search, for the same process running in a series of hosts, consuming constantly network resources. If this process is not known, this observation might indicate that the machines running the specific process are suspicious for participating to a BotNet.

Application	Background Behavior
BitTorrent	A host becomes a seeder
File Sharing	Routing/Downloads/ Uploads are served
Peer-to-Peer	A host becomes a relay
Web 2.0	Polling in periodic intervals for Server data
Social Utilities	Polling in periodic intervals for Server data

Table 2: Application network profiles, when they are not explicitly used.

In general, logging process network activity to a file, and aggregating the information to a central place, may substantially assist in spotting security issues. Analyzing the collected information might not be trivial, but we believe that there are some practical heuristics that may be applied and reveal anomalous host behavior.

Portability

Our implementation of *Topnet* is currently tested on a number of Linux distributions. Though, in the near future, we plan to port *Topnet* to various different UNIX flavors, as well as Windows. *Topnet* is based on two widely used tools. libpcap is available and supported on all UNIX flavors and also on Windows systems. The source code of netstat is available for Linux, and the same information can be retrieved in Windows using the GetTcpTable and GetUdpTable MSDN functions.[6] We are currently exploring ways to achieve the functionality given by netstat on other UNIX flavors,

[6]http://msdn.microsoft.com/en-us/library/aa366071(VS.85).aspx

using tools like lsof [6] and libproc [2], in order to make *Topnet* more portable.

Kernel Space. The network has become a perpetually used resource of every system. Our implementation aim was to achieve per-process network monitoring on a host machine without changing the OS kernel. However, we believe that this data should be provided by the OS kernel, so that it would be completely accurate and more efficient and would also allow for a new generation of monitoring tools with the network as an inherently monitored resource.

Deployment. Finally, one of our major concerns is the deployment of the tool in large scale networks, that have different purposes. We plan to install the tool in enterprise, research and academic networks. Each different network class has specific properties. Deploying *Topnet* in such environments will give us a better understanding of the tool's potential usage. It will also stress the utility in real environments that exhibit realistic system and network load.

Conclusion

In this paper, we presented a process-centric approach to network resource attribution for UNIX computers. We described the design and implementation of *Topnet*, an extension of the familiar top utility. While top focuses on process attributes like CPU and memory usage, *Topnet* provides information about network usage as well.

In contrast to prior approaches, we implemented *Topnet* without needing any kernel modifications, using only user-level functionality based on the libpcap library. Our evaluation suggests that the performance overhead incurred by *Topnet* is reasonable in most situations.

Overall, *Topnet* provides a familiar and accurate way to attribute network traffic to individual processes, enabling users to gain a process-centric view of the network traffic in their systems.

Acknowledgments

This work was supported in part by the project CyberScope, funded by the Greek Secretariat for Research and Technology under contract number PENED 03ED440. Elias Athanasopoulos is also funded by the Ph.D. Scholarship Program of Microsoft Research Cambridge. We are grateful to our anonymous reviewers for their thoughtful and valuable feedback. We also want to thank our shepherd, Dave Plonka, for his collaboration in improving the paper.

Author Biographies

Antonis Theocharides received his Bachelor in Computer Science from the University of Crete in 2008. He is continuing his studies at the University College of London. He can be reached at atheohar@gmail.com .

Demetres Antoniades received his M.Sc. and B.Sc. degrees in Computer Science from the University of Crete, in 2005 and 2007 respectively. He is currently a Ph.D. candidate in Computer Science in the same university. Since 2004, he is also with the Institute of Computer Science, FORTH, Crete, where he currently works in the Distributed Computing Systems Lab. His main research interests include network monitoring and traffic classification.

Michalis Polychronakis is a Ph.D. candidate in the Computer Science department at the University of Crete, Greece. He received the M.Sc. and B.Sc. degrees in Computer Science from the same university in 2003 and 2005, respectively. Since 2002, he is also with the Institute of Computer Science, FORTH, Crete, where he currently works in the Distributed Computing Systems Lab. His main research interests include systems and network security, intrusion detection, and network monitoring.

Elias Athanasopoulos received a B.Sc. in Physics from the University of Athens and an M.Sc. in Computer Science from the University of Crete, in 2004 and 2006 respectively. He is currently a Ph.D. candidate in Computer Science with the University of Crete. He has published in ACNS, CMS, EC2ND, IWSEC and ISC. He is a Research Assistant with FORTH and he has received a Ph.D. scholarship from Microsoft Research (Cambridge) for the period 2008-2011. He interned in Microsoft Research during the summer of 2007.

Prof. Evangelos Markatos (markatos@ics.forth.gr) received his diploma in Computer Engineering from the University of Patras in 1988, and the M.S. and Ph.D. degrees in Computer Science from the University of Rochester, NY in 1990 and 1993 respectively. Since 1992, he is an associated Researcher at the Institute of Computer Science of the Foundation for Research and Technology – Hellas (ICS-FORTH) where he is currently the founder and head of the Distributed Computing Systems Laboratory. He conducts research in several areas including distributed and parallel systems, the World-Wide Web, Internet Systems and Technologies, as well as Computer and Communication Systems Security. He is currently the project manager of the LOBSTER and NoAH projects, both funded in part by the European Union and focusing on developing novel approaches to network monitoring and network security. Since 1992, he has also been affiliated with the Computer Science Department of the University of Crete, where he is currently a full Professor.

Since 2001 Professor Markatos has been the head of the W3C (World Wide Web Consortium) Office in Greece, one of only 17 such offices around the world. Since 2005, he serves as a member of the Permanent Stakeholders Group of ENISA, the European Network and Information Security Agency.

Bibliography

[1] *ATOP – System & Process Monitor*, http://www.atcomputing.nl/Tools/atop/home.html .

[2] *libproc*, http://opensolaris.org/os/community/observability/process/libproc/ .

[3] *nuttcp Official site*, http://www.lcp.nrl.navy.mil/nuttcp/ .

[4] *RFC 4898 – TCP Extended Statistics MIB*, http://www.ietf.org/rfc/rfc4898.txt .

[5] *The netfilter.org Project*, http://www.netfilter.org .

[6] Abell, V., *lsof – LiSt Open Files*.

[7] Anagnostakis, K. G., S. Ioannidis, S. Miltchev, J. Ioannidis, M. B. Greenwald, and J. M. Smith, "Efficient Packet Monitoring for Network Management," *Proceedings of the Eighth IFIP/IEEE Network Operations and Management Symposium (NOMS)*, pp 423-436, April, 2002.

[8] Antoniades, D., M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ubik, and A. Øslebø, "Appmon: An Application for Accurate per Application Traffic Characterization," *Proceedings of IST Broadband Europe 2006 Conference*, December, 2006.

[9] Bernaille, L., I. Akodkenou, A. Soule, and K. Salamatian, "Traffic Classification on the Fly," *ACM SIGCOMM Computer Communication Review*, Vol. 36, Num. 2, pp. 23-26, 2006.

[10] Bos, H., W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis, "FFPF: Fairly Fast Packet Filters," *Proceedings of OSDI'04*, 2004.

[11] Check Point Software Technologies Ltd., *ZoneAlarm*, http://www.zonealarm.com/store/content/home.jsp .

[12] Cisco Systems, *Cisco Network Analysis Module Software*, http://www.cisco.com/en/US/products/sw/cscowork/ps5401/index.html .

[13] Constantinou, F. and P. Mavrommatis, "Identifying Known and Unknown Peer-to-Peer Traffic," *Proc. of Fifth IEEE International Symposium on Network Computing and Applications*, pp. 93-102, 2006.

[14] Crotti, M. and F. Gringoli, "Traffic Classification Through Simple Statistical Fingerprinting," *ACM SIGCOMM Computer Communication Review*, Vol. 37, Num. 1, pp. 5-16, 2007.

[15] Crotti, M., F. Gringoli, P. Pelosato, and L. Salgarelli, "A Statistical Approach to IP-level Classification of Network Traffic," *ICC'06, IEEE International Conference on Communications*, 2006.

[16] Deri, L., *PF_RING* http://www.ntop.org/PF_RING.html .

[17] Deri, L. and S. Suin, "Ntop: Beyond Ping and Traceroute," *Proceedings Tenth IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, pp. 271-283.

[18] Deri, L. and S. Suin, "Effective Traffic Measurement Using ntop," *Communications Magazine, IEEE*, Vol. 38, Num. 5, pp. 138-143, 2000.

[19] Dischinger, M., A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing Residential Broadband Networks," *IMC '07: Proceedings of the Seventh ACM SIGCOMM Conference on Internet Measurement*, pp. 43-56, 2007.

[20] Estan, C., S. Savage, and G. Varghese, "Automatically Inferring Patterns of Resource Consumption in Network Traffic," *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 137-148, 2003.

[21] Fink, G., V. Duggirala, R. Correa, and C. North, "Bridging the Host-Network Divide: Survey, Taxonomy, and Solution," *Proceedings of the 20th Conference on Large Installation System Administration Conference*, pp. 247-262, 2006.

[22] Fink, G., P. Muessig, and C. North, "Visual Correlation of Host Processes and Network Traffic," *Visualization for Computer Security (VizSec)*, 2005.

[23] Goldszmidt, G. and Y. Yemini, "Distributed Management by Delegation," *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)*, pp. 333-340, 1995.

[24] Hughes, E. and A. Somayaji, "Towards Network Awareness," *Proceedings of the 19th Large Installation System Administration Conference (LISA '05)*, pp. 113-124.

[25] IBM, *Aurora – Network Traffic Analysis and Visualization*, http://www.zurich.ibm.com/aurora .

[26] Inoue, H., D. Jansens, A. Hijazi, and A. Somayaji, "NetADHICT: A Tool for Understanding Network Traffic," *Proceedings of the 21st Conference on Large Installation System Administration*, 2007.

[27] Insolvibile, G., "Kernel Korner: The Linux Socket Filter: Sniffing Bytes Over the Network," *The Linux Journal*, p. 86, June, 2001.

[28] Ioannidis, S., K. G. Anagnostakis, J. Ioannidis, and A. D. Keromytis, "xPF: Packet Filtering for Low-Cost Network Monitoring," *Proceedings of the IEEE Workshop on High-Performance Switching and Routing (HPSR)*, pp. 121-126, May, 2002.

[29] Karagiannis, T., A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "File-sharing in the Internet: A Characterization of P2P Traffic in the Backbone," *University of California, Riverside, USA, Tech. Rep*, 2003.

[30] Karagiannis, T., A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is P2P Dying or Just Hiding," *IEEE Globecom*, 2004.

[31] Karagiannis, T., A. Broido, and M. Faloutsos, "Transport Layer Identification of P2P Traffic," *Proceedings of the Fourth ACM SIGCOMM Conference on Internet Measurement*, pp. 121-134, 2004.

[32] Karagiannis, T., K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel Traffic Classification in

the Dark,” *ACM SIGCOMM Computer Communication Review*, Vol. 35, Num. 4, pp. 229-240, 2005.

[33] Lakkaraju, K., W. Yurcik, and A. Lee, “NVisionIP: Netflow Visualizations of System State for Security Situational Awareness,” *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pp. 65-72, 2004.

[34] LeFebvre, W., “Kernel Mucking in top,” *LISA ’94: Proceedings of the Eighth USENIX Conference on System Administration*, pp. 47-56, 1993.

[35] Malan, G. R. and F. Jahanian, “An Extensible Probe Architecture for Network Protocol Performance Measurement,” *Proceedings of the ACM SIGCOMM ’98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 215-227, 1998.

[36] McCanne, S. and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture,” *Proceedings of the Winter 1993 USENIX Conference*, pp. 259-270, January, 1993.

[37] McCanne, S., C. Leres, and V. Jacobson, *libpcap*, Lawrence Berkeley Laboratory, Berkeley, CA, http://www.tcpdump.org/ .

[38] McPherson, J., K. Ma, P. Krystosk, T. Bartoletti, and M. Christensen, “PortVis: A Tool for Port-Based Detection of Security Events,” *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pp. 73-81, 2004.

[39] Moore, A. and K. Papagiannaki, “Toward the Accurate Identification of Network Applications,” *Proceedings: Passive And Active Network Measurement: Sixth International Workshop, PAM 2005*, March 31-April 1, 2005.

[40] Objective Development Software GmbH, *Little Snitch 2*, http://www.obdev.at/products/littlesnitch/index.html .

[41] Oetiker, T., “MRTG – The Multi Router Traffic Grapher,” *Proceedings of the 12th USENIX Conference on System Administration*, pp. 141-148, 1998.

[42] Papagiannaki, K., M. D. Yarvis, and W. S. Conner, “Experimental Characterization of Home Wireless Networks and Design Implications,” *INFOCOM*, 2006.

[43] Plonka, D., “Flowscan: A Network Traffic Flow Reporting and Visualization Tool,” *Proceedings of the USENIX Fourteenth System Administration Conference LISA XIV*, 2000.

[44] Polychronakis, M., E. Markatos, K. Anagnostakis, and A. Øslebø, “Design of an Application Programming Interface for IP Network Monitoring,” *Network Operations and Management Symposium (NOMS 2004, IEEE/IFIP)*, pp. 483-496 2004.

[45] Sen, S., O. Spatscheck, and D. Wang, “Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures,” *Proceedings of the 13th International Conference on World Wide Web*, pp. 512-521, 2004.

[46] Trimintzios, P., M. Polychronakis, A. Papadogiannakis, M. Foukarakis, E. Markatos, and A. Øslebø, “DiMAPI: An Application Programming Interface for Distributed Network Monitoring,” *Proceedings of the Tenth IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006.

[47] Warren, P. and C. Lightfoot, *iftop*, http://www.ex-parrot.com/pdw/iftop/ .

[48] Wood, P., *MMAP libpcap*, http://public.lanl.gov/cpw/ .

[49] Yin, X., W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju, “VisFlowConnect: Netflow Visualizations of Link Relationships for Security Situational Awareness,” *Proceedings of the 2004 ACM workshop on Visualization and Data Mining for Computer Security*, pp. 26-34, 2004.

[50] Zuev, D. and A. Moore, “Traffic Classification Using a Statistical Approach,” *Proccedings: Passive And Active Network Measurement: Sixth International Workshop, PAM 2005*, March 31-April 1, 2005, 2005.

Fast Packet Classification for Snort by Native Compilation of Rules

Alok Tongaonkar, Sreenaath Vasudevan, and R. Sekar – Stony Brook University

ABSTRACT

Signature matching, which includes packet classification and content matching, is the most expensive operation of a signature-based network intrusion detection system (NIDS). In this paper, we present a technique to improve the performance of packet classification of Snort, a popular open-source NIDS, based on generating native code from Snort signatures.[1] An obvious way to generate native code for packet classification is to use a low-level language like C to access the contents of a packet by treating it as a sequence of bytes. Generating such low-level code manually can be cumbersome and error prone. Use of a high-level specification language can simplify the task of writing packet classification code. Such a language needs features that minimize the likelihood of common errors as errors in the packet processing code can crash the intrusion detection system, which may leave it open to attacks.

To overcome these problems, we use a rule-based specification language with a type system for specifying the structure and contents of packets. The compiler for the specification language generates C code for packet classification. This code can be compiled into native code using a C-compiler and loaded into Snort as shared library. Our experiments using real and synthetic traces show that the use of native code results in a speedup of the packet classification of Snort up to a factor of five.

Introduction

Recent years have seen a rapid escalation of security threats making Network Intrusion Detection Systems (NIDS) critical components of modern network infrastructure. A NIDS inspects each packet for a match against a large signature set. The NIDS must work at near wire speed to be effective in an online analysis mode. Typical NIDS perform a number of operations upon receiving a packet like buffering the packet, matching the packet against signature set, and logging packets or alerts. The performance of each of these operations affects the performance of the system as a whole. Even so, the signature matching component remains one of the most important factors that determines the performance of these systems.

Generally, the signature matching used in IDS consists of two distinct operations: i) *packet classification*, which involves examining the values of packet header fields, and ii) *deep packet inspection*, in which the packet payload is matched against a set of predefined patterns. According to [3], packet classification and deep packet inspection are the most expensive parts of Snort (a popular open source IDS) [10], accounting for 21% and 31% of the execution time. The signatures for Snort-like systems are usually specified using simple rule-based language. Typically, the IDS uses an interpreter to check whether any rule matches an incoming packet.

A lot of research has focused on improving the performance of signature matching component of Snort. Most of the research has focused on deep packet inspection which involves string matching and regular expression matching. Current versions of Snort (i.e., starting with version 2.0) use an improved detection engine that matches strings in parallel. Snort uses many efficient and high-speed string matching algorithms like Aho-Corasick [1] and Wu-Manber [13] to match strings in parallel. If the string match succeeds for certain rules, then the packet header fields in thoses rules are checked sequentially. Snort uses Perl Compatible Regular Expression (PCRE) library for checking regular expressions. The regular expressions are also checked sequentially for the rules for which string matching has succeeded.

In this paper, we look at the problem of improving the performance of packet classification used in Snort. We use a technique based on generating native code for packet classification. Use of native compilation to speed up programs is a common technique used in programming language domain. For example, Johansson and Jonsson [5] have shown how simple native compilation can increase the speed of Erlang programs. Native code for packet classification can be generated by writing C code which understands the grammar of packet formats, and performs necessary byte-order and alignment adjustments. However, writing such C code is cumbersome and error-prone. Moreover, maintaining such low-level code is tedious as making even a small change to the rule set may involve making lot of changes in the C code. Making frequent large-scale changes to the low-level C code makes it difficult to have high confidence that the code is performing signature matching correctly.

[1]This research is supported in part by ONR grant N00014 0710928 and NSF grant CNS-0627687.

To overcome this problem, we use a high-level specification language with a special type-system for packet processing to specify Snort rules. The compiler for the language generates C code which can be compiled using common C compiler like GCC to get native code for packet classification. We provide a translator to convert existing Snort rules to a specification in this language. This way system administrators need not learn the high-level specification language and can continue specifying rules in the Snort rule language.

In the Snort Overview section, we discuss Snort's detection engine. The Packet Classification Code section discusses different approaches to generate packet classification code. We describe the type system used to generate native code in the Specification Language section. The implementation details are provided in the Implementation section which is followed by Evaluation, Related Work, and the Conclusion.

Snort Overview

In this section we discuss the rule language of Snort and the signature matching scheme used in Snort.

Snort Language

Snort uses a simple rule-based language to specify signatures. Snort signatures are written in a configuration file which is read when Snort starts up. A Snort signature file consists of variable declarations and rules. The variable declarations are similar to typedefs in C; the value of the variable is substituted in the rules for signature matching. The rules themselves consists of a *rule header* and a *rule body*.

The rule header consists of *action*, *protocol*, *ip addresses*, *ports*, and *direction operator*. Rule actions specify the action like logging or alerting that Snort should perform when a rule matches a packet. Each rule is applicable to packets belonging to a particular protocol like TCP, UDP, ICMP, or IP. For TCP and UDP rules, the header specifies the source and destination ip addresses and port fields for which the rule is to be applied. Specifying *any* for one of these fields means that the field in the rule matches for any value in a packet. The fields to the left of the direction operator (→) are the source fields, while the ones on the right hand side are for the destination. An alternative operator, called bidirectional operator (<>), indicates that the rule is to be applied to both directions of the flow. Consider the following variable declaration and rule:

```
var internal_host 192.168.2.0/24
alert tcp $internal_host any -> 192.168.1.1 80
```

"internal_host" is a variable whose value is the host address 192. 168. 2. 0 with subnet mask of 24 bits. So any host with this subnet address matches internal_host variable. This rule generates an alert when it sees a tcp packet from any port on an internal host to host 192. 168. 1. 1 on port 80.

Rule body consists of rule options which belong to one of the following categories: i) *meta-data* options provide information about the rule but are not used in signature matching operation, ii) *payload* options are concerned with tests for deep packet inspection, iii) *non-payload* options specify other tests including tests on packet header fields, and iv) *post-detection* options specify some triggers which are fired when a rule matches a packet. Consider the rule body appended to the previous Snort rule:

```
var internal_host 192.168.2.0/24
alert tcp $internal_host any ->
  192.168.1.1 80 (msg:''web-attack'';
  ttl: 5; content: ''abc'';
  logto: ''logfile'';)
```

In this rule, *msg* is a meta-data option that specifies the message to be generated when a packet matches this rule. *logto* is a post-detection option that specifies the file to be used for logging. *content* is a payload option which means that the rule is matched by a packet only if the payload contains the string "abc". Further, the packet has to satisfy the constraint that *ttl* field value is equal to 5 for the rule to match.

Snort Detection Engine

Starting with version 2, Snort uses an improved detection engine for matching signatures. The rules are first grouped based on the protocol field. Thus, all rules are put in one of the groups corresponding to TCP, UDP, ICMP, and IP. The rules are further grouped based on certain fields – *source* and *destination ports* for TCP and UDP, *type* for ICMP, and *protocol* for IP rules. For each group, the strings specified by rules in the group are combined to form an Aho-Corasick automaton. The Aho-Corasick automaton is a deterministic finite automaton that can be used to match the payload against multiple strings in parallel.

When Snort receives a packet, it identifies the group to which the packet belongs. Then the payload of the packet is matched against the Aho-Corasick automaton corresponding to that group. Aho-Corasick algorithm identifies all the rules whose *content option* is matched. For each of these rules, an interpreter checks whether the other payload and non-payload options are satisfied by the packet. If all the options of a rule are satisfied, then a match is announced for that rule.

Packet Classification Code

Signature matching operation consists of two main components: i) *packet classification*, and ii) *deep packet inspection*. Packet classification is the problem of identifying the rules matching a packet based on the values in the packet header fields. The performance of packet classification can be improved by using native code instead of interpreted code. Native code can be generated from packet classification code written in a low-level language like C. The most straight forward way to write such code is by treating the packet as a sequence of bytes. There are many problems with this approach. To access any field of the packet, the offset of that field from the start of the byte sequence has to be calculated. This way of accessing fields with offset calculations has many potential pitfalls. For example, to access the source port field of tcp header, one needs to first ensure that the packet is a tcp packet. The offset for tcp source port depends on the length of the variable-length options field of ip header. Also, the bytes at those offsets need typecasting to `unsigned short` and conversion to the host order. It is clear that writing such code is very tedious and error-prone.

A better approach is to overlay the packet header structure on the byte sequence and then access packet header fields as fields of the structure. Even this approach does not solve the problem completely due to the presence of variable length fields and the need to perform protocol decoding before accessing any field. Another approach is to use a special language developed explicitly for packet processing. Such a language can have a hand-crafted type checker for particular network protocols or have a generic type checker that supports different network protocols. In the former approach, the packet structure for supported protocols are hard coded into the compiler. This approach is very rigid and supporting new protocols requires modification to the compiler. We use the latter approach which is more flexible and extensible.

In [11] we presented a special type system that can capture packet structures while providing the capabilities to dynamically identify packet types at runtime. In the next section we describe the features of the high-level language that uses that type system.

Specification Language

The language for specifying packet classifiers is rule-based. Specifications consist of variable and type declarations, followed by a list of rules. In the following sections we describe each of these components of specifications.

Packet Structure Description

The structure of the packets has to be specified using packet type declarations before specifying the rules. The syntax of type declaration for packets is similar to that of the C-language. For example, Listing 1 describes an Ethernet header.

The nested structure of protocol header can be captured using a notion of inheritance. For example, an IP header can be considered as a sub-type of Ethernet header with extra fields to store information specific to IP protocol. The specification language permits multilevel inheritance to capture protocol layering. Inheritance is augmented with constraints to capture conditions where the lower layer protocol data unit (PDU) has a field identifying the higher layer data that is carried over the lower layer protocol. For instance, IP header derives from Ethernet header only when `e_type` field in the Ethernet header equals `0800h`; see Listing 2.

To capture the fact the same higher layer data may be carried in different lower layer protocols, the language provides a notion of disjunctive inheritance. The semantics of the disjunctive inheritance is that the derived class inherits fields from exactly one of the possibly many base classes. Listing 3 shows a specification that IP may be carried within an Ethernet or a token ring packet.

```
#define ETHER_LEN 6
struct ether_hdr {
    byte e_dst[ETHER_LEN];       /* Ethernet destination address */
    byte e_src1[ETHER_LEN];      /* Ethernet source address */
    short e_type;                /* Protocol of carried data */
};
```

Listing 1: Ethernet header description.

```
#define ETHER_IP 0x0800
struct ip_hdr : ether_hdr with e_type == ETHER_IP {
    bit          version[4];     /* IP Version */
    bit          ihl[4];         /* Header Length */
    byte         tos;            /* Type Of Service */
    short        tot_len;        /* Total Length */
    ...
    short        check_sum;      /* Header Checksum */
    unsigned int s_addr;         /* Source IP Address Bytes */
    unsigned int d_addr;         /* Destination IP Address Bytes */
};
```

Listing 2: IP header with inherited Ethernet header.

```
struct ip_hdr : (ether_hdr with e_type == ETHER_IP) or
                (tr_hdr with tr_type == TOKRING_IP) {
    ...
}
```

Listing 3: Headers for Ethernet and Token Ring.

We can declare a variable of type *ether_hdr* and access various packet fields by using the fields in the respective structure. For example, to access the source port of a tcp packet, we declare a variable corresponding to packets as follows:

```
ether_hdr p;
```

Now, *p.tcp_sport* stands for source port of a tcp packet.

Rules

The rules are of the form "*cond* → *actn*", where *actn* specifies the action to be taken on a packet that matches the condition *cond*. The *cond* is a conjunction of tests on packet fields. The language supports various tests like equality, disequality, and inequality along with bit-masking operations on packet fields. A packet matches a rule if all tests in the rule succeed. If multiple rules match at the same time, actions associated with each rule are launched. Consider the rule in Listing 4. The first test in the rule is equivalent to checking whether the source address of the packet belongs to 192. 168. 2. 0/24. Here, `0xc0a80200` is the hex representation of 192. 168. 2. 0 and `0xffffff00` corresponds to the 24-bit subnet mask. This rule further checks that destination address is `0xc0a80100` (192. 168.1.0) and destination port is 80. This rule is equivalent to the Snort rule shown in the Snort Language section.

The compiler generates a C function for this rule set which takes a network packet (as byte sequence) as input, performs the matching, and returns the rules that match. The packet matching code contains appropriate offset calculation, byte alignment, and order adjustment code.

Constraint Checking

An important requirement for the language to be type safe is that the constraints must hold before the fields corresponding to a derived type are accessed. Note that at compile time the actual type of the packet is not known. For example, a packet on an Ethernet interface must have the header given by *ether_hdr*. But it is not known whether the packet carries an ARP or an IP packet. So the constraint associated with *ip_hdr* must be checked at runtime before accessing the IP-relevant fields. Similarly, before accessing TCP relevant fields, the constraints on *tcp_hdr* must be checked. Furthermore, the constraints on *ip_hdr* must be checked before checking constraints on *tcp_hdr*.

The compiler automatically inserts the appropriate constraints before each field test using the packet structure specification (described in the Packet Structure Description section). For example, the compiler automatically transforms the previous rule to that shown in Listing 5.

Here, the test to the left of **:** is the precondition that needs to be satisfied before the test on the right can be performed. For the first test in the rule, the compiler figures out that *s_addr* is a field in the *ip_hdr* structure. So it adds the constraint on *ip_hdr*, i.e., *e_type* ≡`0x800`, to this test as precondition. Before accessing tcp fields, the constraints on *ip_hdr* and *tcp_hdr* need to be satisfied. As shown in the test on *tcp_dport* in this example, the compiler adds these constraints as a list. Note that the compiler adds these constraints in the order defined by the inheritance chain of packet structures.

Implementation

We implemented a Perl based translator for converting Snort rules into a specification for our language. The translator generates the packet structure specification and generates a rule in the specification for each rule in a Snort rule file. So there is a one-to-one correspondence between the rules in the Snort rules file and the rules in our specification file. The rules in our specification contain only the tests on packet header fields. The other tests in the rules are checked by using the detection engine of Snort itself.

For each non-payload detection option of Snort rules we generate the corresponding packet field test in our language. For example, consider the following rule in Snort,

```
alert tcp $EXTERNAL_NET any
     -> INTERNAL_NET $HTTP_PORT
     (..., ttl: 5; ...)
```

This rule generates alerts for tcp packets with *ttl* field of 5 from external network to internal network on

```
R1: (p.s_addr & 0xffffff00 == 0xc0a80200) &&
    (p.d_addr == 0xc0a80100) &&
    (p.tcp_dport == 80) -> alert(R1);
```

Listing 4: Sample packet matching rule.

```
R1: (p.e_type == 0x800):(p.s_addr & 0xffffff00 == 0xc0a80200) &&
    (p.e_type == 0x800):(p.d_addr == 0xc0a80100) &&
    (p.e_type == 0x800):(p.proto == 0x11):(p.tcp_dport == 80) -> alert(R1);
```

Listing 5: Transformed R1.

```
R1: (p.proto == tcp) && (p.s_addr == $EXTERNAL_NET) &&
    (p.d_addr == $INTERNAL_NET) && (p.tcp_dport == $HTTP_PORT) &&
    (p.ttl == 5) -> alert(R1)
```

Listing 6: Alerts for tcp packets with *ttl* field of 5.

port for http (80). The corresponding rule in our specification language is shown in Listing 6.

We use our compiler to compile the Snort rules in our specification format into C code. The compiler generates a backtracking automaton that matches each rule sequentially. Then it generates the C-code for matching this automaton in a straight-forward way using *if-then-else* branching. We use C compiler like *gcc* to generate native packet classification code in the form of a shared library. So to update the rules, all that one needs to do is to compile the rules offline and then reload the shared library. We note that this approach is no more disruptive than that of Snort where the rules need to be re-read and recompiled.

We load the shared library containing the packet classification code when Snort starts up. At runtime, when a packet is delivered to Snort by *pcap library*, we pass on the packet to the shared library. The shared library matches the packet against all the rules and returns the rules that match. At this point control is transferred to the default Snort detection engine. From this point on, the usual Snort processing (like logging) is performed on the packet.

We note that using this approach does not modify the behavior of Snort. In particular, for any packet the modified Snort matches the same rules as the original Snort. This is because we are just changing the way packet classification is performed without changing the actual tests in a rule.

Evaluation

We evaluated the effectiveness of the proposed technique using Snort 2.6.1.5. Our experiments were performed on a system with 3.06 GHz Intel Xeon processor and 3 GB memory, running Fedora Core 5 (Linux kernel 2.6.15). To understand the impact of native compilation of Snort rules we used the default signatures that come with two different versions of Snort – Snort-1.8 and Snort-2.6. Snort-1.8 uses a slower scheme for matching packet fields where each rule is checked sequentially without first grouping the rules. To compare the performance of native code for packet classification, we generate native code for matching similar to the way used in Snort-1.8. This is a very simple way of matching where a packet is matched against all the rules for the protocol that it belongs to. For example, a tcp packet will be matched against all tcp rules. On the other hand, in Snort-2.6 it will be matched against only the tcp rules whose source and destination port values are compatible with the values in the packet.

We converted the signatures to a specification for our language as described in implementation section by combining rules which differed only in payload detection options. So we were left with around 300 unique rules on packet header fields for Snort-1.8 rule set and 600 rules for Snort-2.6 rule set. We ignore the payload options for evaluating the packet classification schemes.

We used two sets of packet traces for measuring runtime performance. The first one consists of all packets captured at the external firewall of a medium-size University laboratory that hosts about 30 hosts. Since the firewall is fully open to the Internet (i.e., the traffic is not pre-screened by another layer of firewalls in the University or elsewhere), the traffic is a reasonable representative of what one might expect a NIDS to be exposed to.

Our packet trace consisted of about 8 million packets collected over a few days. The second one corresponds to 10 days of packets from the MIT Lincoln Labs IDS evaluation data set [8], consisting of 17 million packets. We used Snort-2.6 in offline mode to perform these experiments. In offline mode, Snort reads the packets from a trace file using pcap_loop function call of pcap library. For each packet, pcap_loop calls a callback function.

In Snort, the callback function first decodes the packet and passes it to detection engine, i.e., the component that performs signature matching. We measured the time spent in the callback function by calling `times()` function before and after the call to pcap_ loop.

To get the time spent in detection engine, we first ran Snort without making any call to the detection engine and measured the time spent in the callback function. This is the time spent in reading the packets from file and decoding them. Then we measured the time spent in the callback function for unmodified Snort. The difference in the two measured times gives the time spent in the detection engine.

We modified Snort-2.6 such that after decoding the packet was passed to the function in shared object which performs packet classification. We used the above technique to measure the time spent in this function. To compare the effect of the number of rules on the matching time, we used configuration files with different number of rules and found the packet classification time for the original Snort and the modified Snort that uses native code.

Figures 1 and 2 show the per packet classification time for Snort-1.8 rule set for the first and second packet traces. Even though Snort-2.6 groups the rules based on certain fields for each protocol, it is five times slower for the first packet trace and two times for the second trace when the complete rule sets are used. As expected, the matching time for compiled rules increases linearly with the number of rules but is always better than the interpreted rules by a factor of at least 2. Snort-2.6 stores the rules as some simple data structure in memory. At runtime, this data structure needs to be traversed. This traversal involves many memory accesses. The compiled code on the other hand performs this traversal using simple conditional and unconditional branching intructions. This is

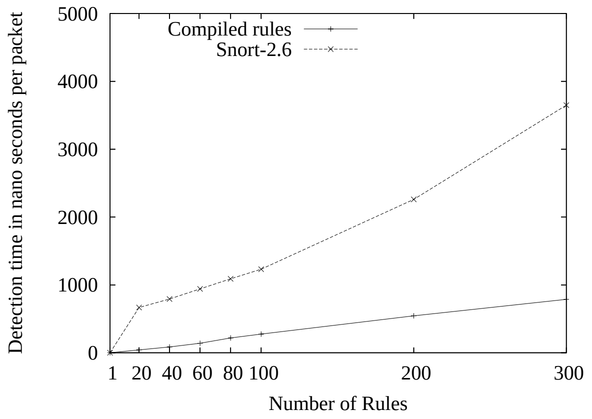


Figure 1: Matching time for Snort-1.8 rules for first trace.

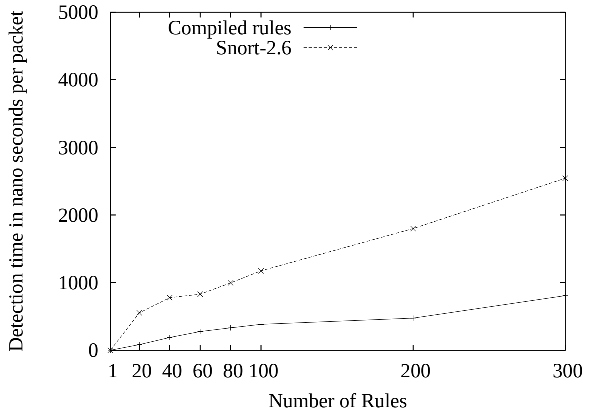


Figure 2: Matching time for Snort-1.8 rules for second trace.

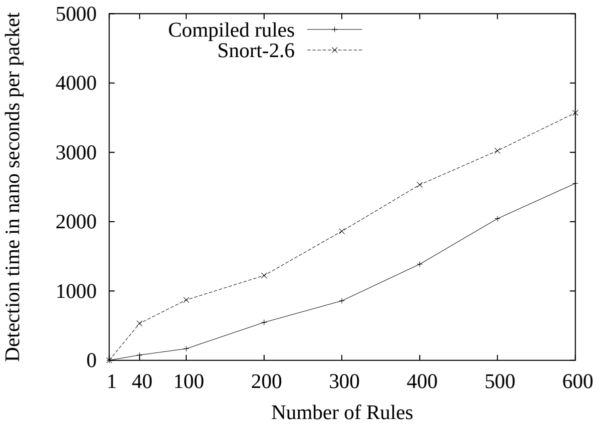


Figure 3: Matching time for Snort-2.6 rules for first trace.

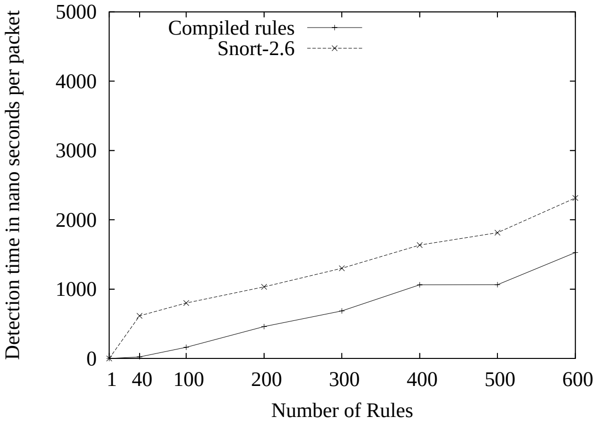


Figure 4: Matching time for Snort-2.6 Rules for second trace.

the main reason for the performance improvement obtained by using native code over interpreted code.

The results for Snort-2.6 (Figures 3 and 4) are qualitatively similar showing that native compilation of packet classification, even with a naive matching scheme, performs about 30% better for the complete rule sets than the interpreted method that uses a more sophisticated scheme.

Related Work

Chandra, et al. developed Packet Types [2], a high-level specification language that provides a type system for packet formats similar to our language. It uses a construct called as *refinement* to capture the notion of inheritance. The inheritance mechanism of Packet Types offers more power than that of our language in that it can capture protocols that use trailers also. Our language trades off this power for simplicity. For the kind of protocols that Snort rules handle, this additional expressive power is not required.

Vern Paxson developed Bro [9] which is another popular open source NIDS. Bro has a powerful policy language that allows the use of sophisticated signatures. Bro comes with a translator to convert Snort rules to Bro signatures [12]. But the semantics of matching differ in Bro and Snort. Unlike snort, which uses signatures that are based on individual packets, Bro performs matching on data streams obtained after packet reassembly. Our goal was to develop a plug-in for Snort that speeds up Snort while preserving the matching semantics.

Kruegel and Toth developed the Snort-NG [6] system, which demonstrated the performance gains achievable by parallelizing the signature matching. They use a two-stage approach where a decision tree is used for packet classification followed by content search. They use an interpreter based approach for packet classification. In that respect, our technique can help them in speeding-up the packet classification.

Previous research [14, 16] has looked at speeding up packet classification using hardware based approach. But these works focus on IP lookup problem, i.e., classifying packets based only on source and destination addresses and ports. Such an approach is useful in routers but not in intrusion detection systems like Snort which use additional fields like *ttl*, *window*, and *tcp flags*.

There has been a lot of research on speeding up the content matching component of Snort. Various software [3, 7] and hardware [15, 4] based solutions have been proposed in this area. In that respect our work can be used in conjunction with these solutions. Our packet classifier can quickly filter out the packets that do not need the expensive content matching operation.

Conclusion

In this paper, we presented a technique for improving the performance of packet classification used in IDS like Snort by using native code while preserving the matching semantics. Generating native code by hand is tedious and error prone. So we used a type system tailored for packet formats to generate type safe code. Our experiments with real and synthetic traces show that use of native code can result in speeding up the packet classification of Snort from 30% up to 80%. In the future, we want to generate native code for content matching.

Author Biographies

All The authors of this paper are members of the Secure Systems Laboratory of Stony Brook and their homepages are accessible on the web from the laboratory page at http://www.seclab.cs.sunysb.edu.

R. Sekar is currently Professor of Computer Science and heads the Secure Systems Laboratory at Stony Brook University. Prof. Sekar's research interests include computer system and network security, software and distributed systems, programming languages and software engineering. He can be reached by email at sekar@cs.sunysb.edu .

Alok Tongaonkar is a Ph.D. student in the CS department at Stony Brook. His main research area is computer security and is currently working on analyzing and optimizing firewalls and intrusion detection systems. He is available at alok@cs.sunysb.edu .

Sreenaath Vasudevan is a M.S. student in the CS department at Stony Brook. Sreenaath does research in the area of computer security. He can be reached via email at sreev@cs.sunysb.edu .

Acknowledgments

We would like to thank Manuel Rivera and Lohit Vijayrenu who helped in writing the translator for Snort rules. We would also like to thank our shephard Brent Hoon Kang for his insightful comments and Rob Kolstad for his help in formatting the final version of the paper.

Bibliography

[1] Aho, A. and M. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Communications of the ACM*, Vol 18, Num. 6, pp. 333-343, 1975.

[2] Chandra, Satish and Peter J. McCann, "Packet Types," *Second Workshop on Compiler Support for Systems Software (WCSSS)*, May, 1999.

[3] Fisk, M. and G. Varghese, *Fast Content Based Packet Handling for Intrusion Detection*, 2001.

[4] Jacob, Nigel and Carla Brodley, "Offloading IDS Computation to the GPU," *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference*, IEEE Computer Society, pp. 371-380, 2006.

[5] Johansson, Erik and Christer Jonsson, *Native Code Compilation for erlang*, Technical Report, 1996.

[6] Kruegel, Christopher and Thomas Toth, "Using Decision Trees to Improve Signature-Based Intrusion Detection," *6th Symposium on Recent Advances in Intrusion Detection (RAID)*, 2003.

[7] Kumar, Sailesh, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner, "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection," *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 339-350, ACM, 2006.

[8] MIT Lincoln Labs, *DARPA Intrusion Detection Evaluation*, 1999.

[9] Paxson, V., "Bro: A System for Detecting Network Intruders in Real-Time," *USENIX Security*, 1998.

[10] Roesch, Martin, "Snort – Lightweight Intrusion Detection for Networks," *13th Systems Administration Conference*, USENIX, 1999.

[11] Sekar, R., Y. Guang, S. Verma, and T. Shanbhag, "A High-Performance Network Intrusion Detection System," *ACM Conference on Computer and Communications Security*, pp. 8-17, 1999.

[12] Sommer, R. and V. Paxson, "Enhancing Byte-Level Network Intrusion Detection Signatures with Context," *ACM CCS*, 2003.

[13] Wu, S. and U. Manber, "A Fast Algorithm for Multi-Pattern Searching," *Technical Report TR-94-17*, 1994.

[14] Yu, Fang and Randy Katz, "Efficient Multi-Match Packet Classification with TCAM," *High Performance Interconnects*, 2004.

[15] Yu, Fang, Randy H. Katz, and T. V. Lakshman, "Gigabit Rate Packet Pattern-Matching Using TCAM," *12th IEEE International Conference on Network Protocols*, 2004.

[16] Yu, Fang, T. V. Lakshman, Marti Austin Motoyama, and Randy H. Katz, "SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification," *Symposium on Architectures for Networking and Communications Systems*, 2005.

Sysman: A Virtual File System for Managing Clusters

Mohammad Banikazemi, David Daly, and Bulent Abali – IBM T. J. Watson Research Center

ABSTRACT

Sysman is a system management infrastructure for clusters and data centers similar to the /proc file system. It provides a familiar yet powerful interface for the management of servers, storage systems, and other devices. In the Sysman virtual file system each managed entity (e.g., power on/off button of a server, CPU utilization of a server, a LUN on a storage device), is represented by a file. Reading from Sysman files obtains active information from devices being managed by Sysman. Writing to Sysman files initiates tasks such as turning on/off server blades, discovering new devices, and changing the boot order of a blade. The combination of the file access semantics and existing UNIX utilities such as grep and find that operate on multiple files allow the creation of very short but powerful system management procedures for large clusters. Sysman is an extensible framework and has a simple interface through which new system management procedures can be easily added. We show that by using a few lines of Linux commands system management operations can be issued to more than 2000 servers in one second and the results can be collected at a rate of more than seven servers per second. We have been using Sysman (and its earlier implementations) in a cluster of Intel and PowerPC blade servers containing hundreds of blades with various software configurations.

Introduction

Managing clusters made of hundreds and thousands of servers with various types of storage and network devices consumes a significant amount of system administration resources. Heterogeneity of devices in a cluster or data center and a plethora of command line interfaces, graphical user interfaces (GUI), and web based solutions available for managing different device types have made the management of these systems an ever more challenging task. Ethnographic studies of system administrators have shown deficiencies in current system management tools, including 1) poor situational awareness from having to interface with several management tools, 2) lack of support for planning and rehearsing complex system management procedures, and 3) incomplete functionality requiring system administrators to build their own tools [14]. It is also shown that the ability to automate and script system management functionality is crucial [14].

Currently, there are several monitoring and management solutions for performing system administrative tasks in data centers and clusters [5, 8, 10, 12, 13, 20, 22, 23, 26]. Each of these solutions use a different graphical or web based interface. These solutions vary in how radical they are and how different an approach they require for dealing system administration and monitoring tasks. We discuss several of these solutions in the Related Work section.

In our work we use a simple interface through which complicated system administrative tasks can be automated easily. We aim to design and provide a powerful yet familiar interface on top of which various system administrative tools can be built. In order to make such an interface successful, instead of providing all the commands for performing various tasks on all devices (which is impossible), we design the system such that manufacturers (and developers) can add the required programs and scripts for a given device to the system. We also design the system such that we take advantage of a large set of familiar system administration utilities (such as filesystem utilities that can be easily used for manipulating large number of files). We call our system Sysman.

Sysman is an easily extensible framework. Our work was initiated by the need to manage a large set of IBM BladeCenters containing tens and hundreds of IBM blades in our lab. In our first implementation, we provided agents mainly targeting IBM BladeCenter [19] platform management and basic Linux server management. Our BladeCenter platform management agents leveraged the libraries developed for [16]. However, we designed Sysman such that it can be enhanced by additional scripts and executables provided by the users and third parties, using a simple and well defined interface. No change to the Sysman file system itself is required for such additions.

Sysman provides a unified and simple interface for managing various devices found in clusters and data centers. Sysman provides a /proc-like interface. The Sysman file system is layered over the command line interfaces and other tools available for managing devices, and provides a very simple yet powerful interface for accessing all these tools in a unified manner. The file system semantics enables the use of for example existing UNIX commands such as find, sort, and grep, and script programming to manage large clusters of servers as easily as a single server. Considering that UNIX (and Linux) file system commands are very simple yet can be utilized to perform very complicated tasks operating on many files, Sysman provides a very powerful and easy to use interface for cluster management. Furthermore, since most system administrators are already familiar with UNIX file system commands and writing shell scripts, they will have a very short learning curve with Sysman. Figure 1 contains three examples of how simply Sysman can be used to monitor and/or control a large number of devices.

In addition to collecting data from servers in a cluster, Sysman uses exactly the same interface to monitor and manage various types of devices. This provides an opportunity to integrate various system management domain into one and reduce the system management cost. For a device to be Sysman enabled, it is enough to have a command line interface through which the device can be monitored and managed. Once such an interface exist, simple scripts can be added to the system to provide the basic methods for managing these devices. Furthermore, as use of virtual machines become more widely used, Sysman can be utilized to create, monitor, manage, and destroy them as well.

System management tasks are performed by accessing Sysman files. In particular, all tasks are accomplished by reading or writing a file or a directory look up operation. Sysman is a virtual file system similar to the Linux /proc or /sys file systems. Sysman creates a virtual file system where accessing files in the /sysman directory results in execution of related system management tasks. Sysman virtual file system is typically created on a management server. Each device or system is represented as a directory. Each device (or system) directory contains files through which various components of the device can be managed. Sysman files are not only bytes on disk but contain the active state of managed devices and systems. One can also consider Sysman as a vehicle for providing named pipes [6] where one side of the pipe is connected to a remote device. Sysman is an easily extensible framework. Sysman allows the inclusion of new devices by simply allowing agents and scripts developed for these devices be utilized by Sysman. Here are the contributions of Sysman:

- Sysman simplifies management of HPC clusters and data centers
- Provides a unified interface for managing different system types, servers, storage systems and network devices.
- File system representation of managed systems enables the use of simple and familiar, yet powerful UNIX (and Linux) file system commands for managing hundreds and thousands of systems as easily as a single system
- Sysman can be extended to support arbitrary device types provided that device operations can be represented as file read and write operations
- Since Sysman and its agents run in user-space, developing new extensions is easy.
- Graphical and web based interfaces can be built on top of Sysman

The rest of this paper includes a background followed by the basic architecture of Sysman and advanced design issues. Related work, future work, and conclusions complete the presentation.

Background

The *proc* file system [18] is a virtual file system used in UNIX systems. Information about the system can be obtained by accessing files in the /proc directory. Linux, SUN Solaris, and IBM AIX are among operating systems which support such a virtual file system. Certain system parameters can be configured by writing to files in the /proc directory. The /proc file system, and the similar /sys file system are used for obtaining information and configuring local resources only. Sysman operates on a similar basis, but as opposed to the /proc system, it is used for managing networked systems. Therefore, it can be used to configure a much more diverse group of systems.

Sysman is a virtual file system developed completely in user space through the use of FUSE, *File system in User-space* [4]. FUSE, is a kernel module, which provides a bridge to the kernel interface. FUSE

```
### A simple command for finding all servers which are turned on
find /sysman/systems -name power | xargs grep -l on

### List all the blade chassis whose temperature is above 35 degrees
find . -name 'temperature_mp' | xargs grep "" | sed s/:/" "/g | \
        awk '{ if ($2 >= 35.0) print $1 "temperature is: " $2}'

### A simple script to collect the vmstat of all servers in background
#!/bin/bash
LIST=`find /sysman/systems -name command`
for i in $LIST; do
    echo "vmstat " > $i &
done
```

Figure 1: Examples of Sysman usage.

is now part of the Linux kernel. It provides a simple API, has a very efficient user-space kernel interface and can be used by non privileged users. The path of a typical file system call is shown in Figure 2a. FUSE can intercept file system calls and redirect them to a user-space program for implementing virtual file systems. Several virtual file systems have been developed on top of FUSE. These file systems provide various features, from versioning, to encryption, to simple methods for accessing special devices. A list of file systems built on top of FUSE can be found at [3].

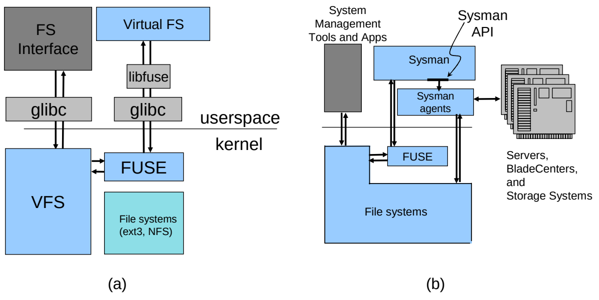


Figure 2: (a) File system call with FUSE and (b) path of typical system administrative tasks with Sysman.

In Sysman, we use FUSE to intercept accesses to Sysman files. While accesses to other files are not affected, certain access types to Sysman files result in information gathering, configuration, and other system administrative tasks. In particular, Sysman intercepts accesses to its directories. In the next section we provide discuss the basic architecture of Sysman.

Basic Architecture

In Sysman, each device is represented as a separate directory with its own files and possibly subdirectories under the /sysman subtree. Figure 2b illustrates the path of typical system administrative tasks. When as part of a system administration task, a file in the /sysman directory is accessed, the file system call is intercepted by FUSE and its processing gets delegated to the user level Sysman program. In the rest of this section we discuss various aspects of Sysman in detail.

Virtual File System Operations

Main system management tasks are performed by either reading from a file in the /sysman directory or by writing into a file in this directory. When a file in /sysman directory is accessed, Sysman determines if there is an agent associated with the requested <file name, file system operation> pair, and executes the matching agent, if it exists, before processing the access as a regular file system read or write. For each <file name, file system operation> pair, the name of the corresponding agent is derived by Sysman. For example, discover_master.write is the agent to execute on write operations to the file discover_master. Sysman stores the agents in a hierarchical manner in a configurable location and therefore given the complete path of a file, the corresponding agent name and location can be easily found. If such an agent exists, it is executed. If not, no agents will be executed. In either case, the file access operation completes as a regular file system operation. That is, if the file system operation is a read from a file, Sysman reads the content of the file, or if the file system operation is a write to a file, it writes into the file.

In general, read operations are used for obtaining information about the status of devices that are being managed. For example, by reading the content of the file *power* in a server directory, one can find out if the server is turned on or not. Sysman supports various information caching models that affect how often system management information is collected. Whenever a file is read, if the information is not already stored in the file, necessary action is taken to obtain the information, store it in the file and then present it to the user. If the information is already available, the behavior depends on the caching model used for that particular file. The cached information can be simply presented to the user or the content can be refreshed first. In certain cases, the information is collected periodically. The caching model is specified in the Sysman configuration file and can be set for a group of files or individual files. We will discuss this feature in more detail in the Special Files and Error Handling section.

On the other hand, writes to certain files result in performing a task on the corresponding device (or component). For example, writing a 1 or the word *on* into a *power* file results in Sysman turning on the server. In cases where the result of the operation is required to be preserved, the result is stored in the file. A subsequent read from the file prints out the result in those cases. Sysman also recognizes certain file and directory names and extensions and performs certain predefined actions in response to accessing them. This feature is discussed in the Special Files and Error Handling section.

The /sysman Directory Hierarchy

The /sysman directory is organized such that each managed device has its own directory. Furthermore, similar devices (e.g., servers with the same type, or blades of an IBM BladeCenter) are listed under one directory. Files in each directory are used for managing the device represented by that directory. Similarly files in parent directories are used for obtaining information about the group a device belongs to. Figure 3 shows parts (not all) of the /sysman directory structure for a BladeCenter system in our lab consisting of eight chassis with 14 blade servers in each chassis (112 blade servers total) and several other servers. The creation of this directory tree is discussed later when we discuss the discovery mechanisms. Note the 8 chassis found under the directory /sysman/systems/chassis/. Each chassis is represented by a directory named after

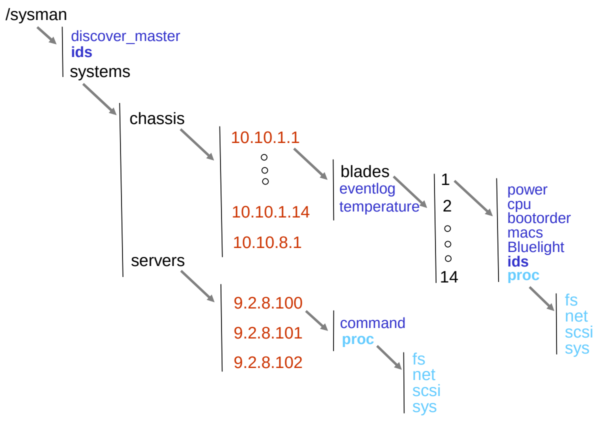


Figure 3: Partial snapshot of a /sysman directory.

the IP address of the chassis management module, for example 10.10.1.1.

In Figure 3, under each chassis subdirectory there is a *blades* directory and two files, *temperature_mp* and *eventlog*, which report the hardware temperature and hardware events common to the chassis, respectively. Blades directory contains up to 14 subdirectories named 1 to 14, each representing a blade server found in the chassis. Reading from files found under the blade directory, reports the blade status and properties. For example, the first command in Figure 4 displays the boot order of blade number 3 in the BladeCenter chassis whose Management Module (MM) IP address is 10.10.1.1. Likewise, for configuring devices, Sysman files may be written to. For example, to change the boot order of the blade, one may execute the second command in Figure 4.

Sysman is configured by a single configuration file. This configuration file specifies among other things the location of the Sysman agents (Figure 5a) and where the FUSE directory is mounted. By default, any access to a Sysman file refreshes the content of the file first. On the other hand, by default accessing a directory does not cause refreshing the directory except for *proc* and *sys* directories.

Discovery

Sysman provides three methods for adding new devices to the system: prompted, automatic, and manual.

The prompted method requires the writing of '1' to the /sysman/discover_master file. Once the file is written to, Sysman searches for devices known to it and appropriate directories and files are created upon discovery of new devices. BladeCenter chassis are discovered through the Service Location Protocol (SLP) [25]. The prompted method is used by issuing a command like the first command in Figure 6. This command causes Sysman to run the discover_master.write script which runs an SLP client program. The SLP client makes broadcast announcements to discover all the BladeCenter chassis on the local subnet. For each chassis discovered, a directory is created in the /sysman/system/chassis directory, with the IP address of the chassis management module as the name of the directory.

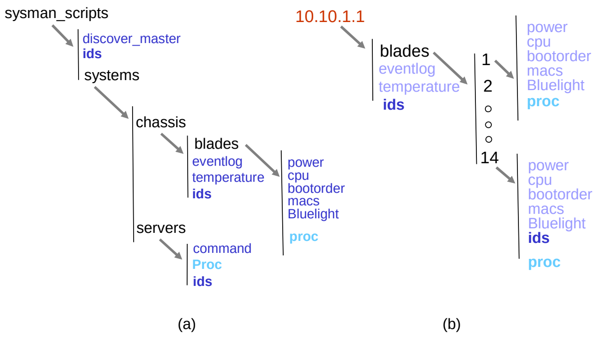


Figure 5: (a) Parts of a Sysman script directory and (b) different ids files.

In the automatic method, Sysman periodically runs the same discovery processes as described above without user intervention. A discovery frequency is specified in the configuration file. When a new device is discovered that contains other devices, discovery is run automatically on the new device as well. In the manual method, the name or IP address of the device (or its management interface), along with the device type is written to the discover_master file. As a result of such a write, corresponding directories and files are created in the /sysman directory. The second command in Figure 6 adds a server to the system.

Sysman File System Security

Sysman uses the existing file system authentication and permission system used in Linux to control access to the files under /sysman. Access to /sysman can be limited to the root user if need be. If the access is not limited to root, regular file system permissions are checked to see if a user can access a file under /sysman. For example, users in a system administration group can be given both read and write access to sysman files, while other users can be limited to read access. In another example, a user or a group can be restricted to accessing only a subtree of /sysman, therefore authorizing those users to manage only a subset of the managed systems. Furthermore, other

```
$ cat /sysman/system/chassis/10.10.1.1/blades/3/bootorder
$ echo "network, floppy, cdrom, harddisk" > bootorder
```

Figure 4: Examples of reading from and writing to Sysman files.

```
$ echo "1" > /sysman/discover_master
$ echo "server 10.10.2.15" > /sysman/discover_master
```

Figure 6: Methods of discovery in Sysman.

available access control lists can be also used for enhanced file security.

The Command File

A generic method for running tasks on a remote server using ssh is provided by writing the command into the *command* file. When a command is written to a file called *command* in the /sysman directory tree, the Sysman executes that command remotely. The result of the remote execution is written into the *command* file. A subsequent read from this file prints out the result of the command execution. The command agent provides a simple method for Sysman users to perform tasks not found under /sysman. This is similar to the C function system(<*string*>), except that the <*string*> runs on multiple systems found under the /sysman tree. For example, to get a list of running processes from all systems, one could execute the sequence of commands in Figure 7.

Advanced Features

In this section we describe some of the more advanced features of Sysman.

Authentication for Access to Managed Systems

Many managed systems and devices require their own authentication method or information. Since Sysman wraps other system management tools, Sysman must support a way to comply with the authentication requirements of the lower level tools and cache credentials. This is done through the use of the *ids* files. The user id and password are provided by the user by writing them into the ids file. These values are then encrypted and kept by Sysman in memory and are **not** written to a file as a security measure. Sysman agents can look up these credentials whenever they need them.

Inheritance and Special Files

Sysman supports inheritance in the /sysman hierarchy. One example of this feature is the *ids* file which contains <userid,password> pairs. In Figure 5b the user id and password for accessing blade 14 is found in the *ids* file under the blade 14 directory. However, blade 1 does not have its own *ids* file. In Sysman, it is not necessary to create an *ids* file for each managed device or system. The user ids and passwords propagate down directory trees for the convenience of the user.

If a system management operation requires a user id and password, but there is no *ids* file in the corresponding directory, the ids of parent directory is searched for such an entry. This process is continued up in the directory tree until an ids file is found. Then the user id and password are retrieved by accessing the *ids* file. (The /sysman/ids file represents the top level *ids* file.)

Special Files and Error Handling

Sysman recognizes some files such as *ids* files as special files. Accesses to special files leads to the execution of predefined tasks by Sysman. Another special file is the *lock* file. In cases where a resource is shared by multiple devices (for example the management controller of several blades) and access to the shared resource needs to be serialized, Sysman provides the required mechanism for locking (by using a file called *lock* in the location corresponding to the shared resource of interest).

Special files are used for handling errors. In Sysman, files ending with *.status* extension are special files. These files follow a simple format:

```
<return_value [return_message]>
```

Sysman and its agents use this file to store the return value of an operation (an integer) and possibly a human readable message indicating what the result is. If the operation succeed the return value will be zero. For example the status of executing a command initiated by writing to the *power* file in a directory will be stored in *power.status* in the same directory.

This feature can be used to implement various error handling methods at Sysman level. That is, Sysman users can write their programs and scripts such that depending on the result of an operation, suitable actions can be taken. Another level of error handling can be performed at the level of agents. Agents used for managing devices can do more than passing the result status by using *.status* files. Agents can themselves be responsive to various error conditions and try to recover from various failures. (Our current implementation uses agents which can respond to failing connections, etc.)

Unified /proc and /sys

Directories called *proc* and *sys* are special directories in Sysman. Sysman consolidates information provided under the /proc and /sys directories of all managed Linux systems. Since these directories change dynamically, their content gets updated by accessing the corresponding remote device, whenever they are accessed. In other words, by default these directories will be refreshed upon access. Additional directories which require the same treatment can be defined in the Sysman configuration file.

API and Extensibility

Sysman supports a simple extensible interface for calling its agents (the underlying scripts and binary executables). Whenever a Sysman file is accessed the name and path of the agent to be executed are derived by Sysman. Then the agent is run and a set of parameters are passed to the agent as options (Figure 8).

```
$ echo "ps -ax" > run_this
$ find /sysman/systems/servers -name command -exec cp run_this {} \;
$ find /sysman/systems/servers -name command -exec cat {} \;
```

Figure 7: Issuing commands to Sysman devices.

Currently, in addition to the complete path and file name of the file being accessed, three options are supported. User id and password (the *-u* and *-p* options) are used when accessing the device requires the use of a user id and password. The *-i* option is used to pass the information provided by user for accessing the file. For example, the content the user wants to write into a file is passed to the agent through the use of this option. As it can be seen, this interface is very simple and flexible. Agents for performing new tasks and/or supporting new devices can be easily developed by following this interface. Once the agent is developed and tested, it can be easily added to the Sysman directory for agents.

```
agent -f file_name [-u user_id]
           [-p password] [-i user_input]
```

Figure 8: Sysman API.

Asynchronous Execution

The main strength of Sysman is that it can be used along with a variety of file system and operating system commands and utilities. Sysman can be utilized in an asynchronous manner simply by accessing Sysman files asynchronously (i.e., in the background). This feature is very useful when hundreds and thousands of devices are being managed by Sysman. Let us consider a case where we want to obtain the CPU utilization (and other information provided by the *vmstat* command) of our compute nodes. If we collect this information node by node it will take a long time before all nodes are contacted. Alternatively by accessing corresponding files in the background as shown in the example below by using the "&" sign we can initiate access to the nodes without waiting for the responses to arrive (Figure 9).

```
## A simple script to collect the vmstat
## of all servers in background
#!/bin/bash
LIST=‘find /sysman/systems -name command‘
for i in $LIST; do
  echo "vmstat " > $i &
done
```

Figure 9: Asynchronous execution of Sysman commands.

Figure 10 shows how long it takes to get this information from up to more than 2000 nodes synchronously and asynchronously. It can be observed that when the operation is done asynchronously, the parallelism in obtaining the information from different nodes results in an order of magnitude reduction in total time.

Related Work

The difficulties in managing clusters containing many physical machines and now hundreds and thousands of virtual machines have led several research and development groups to work on various methods for automating system administrative tasks. Very different approaches have been taken by these groups. In this section we discuss some of the related work.

Plan 9 is a distributed system built at AT&T Bell Laboratories [24] in which all system resources are represented by a hierarchical file system. Files in this file systems are not repositories for storing data on disk. However, resources are accessed using file-oriented read and write calls. The network protocol used for accessing and manipulating system resources is called 9P. Sysman uses a similar method for performing system management tasks.

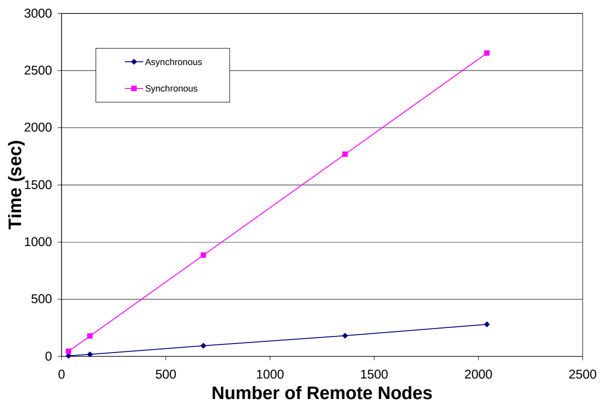


Figure 10: Sysman synchronous and asynchronous execution modes.

Xcpu [22] is a 9P based framework for process management in clusters and grids developed as Los Alamos National Laboratory intended to be a replacement for the bproc system for managing processes across a cluster. The system uses 9p to develop a directory tree to control the scheduling and execution of jobs on a large cluster. Each node in the system is represented by a directory, including subdirectories for all sessions running on the nodes. All processes are run within a session.

The session directory includes the files arch, ctl, exec, and status among others. The arch file can be read indicating the architecture of the node, and the status file reports on the status of the session. The ctl and exec files are used to control the execution of a process in the session. The target image to run on the node is specified by copying the image to the exec file, while the control of the session is controlled by writing various values to the ctl file.

Xcpu uses a similar architecture in representing system resources as directories and files that can be accessed for system management tasks. Configfs [2] is a user-space driven approach for configuring kernel objects. Similar to Xcpu and Sysman, resources are presented in a directory hierarchy and configuration tasks are achieved by accessing files. While Xcpu is mainly used for process management and Configfs is used for configuring kernel objects, Sysman is used for managing a larger and more diverse set of resources.

Nagios [8] is a host and network monitoring application for monitoring *private* services and attributes such as, memory usage, CPU load, disk usage, and Running processes. Nagios watches hosts and services specified to it, and alert the user when things go bad and when they get better. It provides the ability to define event handlers to be run during service or host events. Ganglia [5] is another distributed monitoring system for high-performance computing systems such as clusters and Grids. It uses It uses widely used technologies such as XML, XDR, and RRDtool for data storage and visualization. IBM Cluster Systems Management (CSM) [1] is a management tool for clustered and distributed Intel and PowerPC systems. CSM supports Linux and AIX operating systems and can be used for system deployment, monitoring and management. In particular, CSM is designed to provide the parallelism required for efficient management of clusters made of hundreds of nodes. In contrast with these tools and products, for system administrators familiar with UNIX and UNIX-like environments, Sysman is very simple and can be deployed and used effectively in a very short amount of time.

Usher [21] and MLN [15] aim for providing an extensible framework for managing large clusters and networks. In particular, Usher is a virtual machine management system with an interface whereby system administrators can ask for virtual machines and delegate the corresponding administrative tasks to modular plug-ins. The architecture of Usher is such that new plug-ins can be developed and utilized by users. The goals behind the architecture of Sysman is similar to those of Usher in that they both aim to provide an extensible framework where users can add their own modules (scripts) when need be. Sysman differs from Usher in its scope and the fact that it tries to target wider array of systems (including virtual systems) and devices such as storage and network devices.

Future Work and Conclusions

An early version of Sysman with limited set of features has been available on SourceForge [9]. We are extending our work to support storage devices as well. Currently we have developed several agents for managing IBM DS4000 series storage systems. These agents are used to create LUNs and map them to appropriate host systems. Although not discussed in the current version of the paper, we have completed this work. We are also extending Sysman to monitor and manage networking devices (in addition to those network switches which can be managed through IBM BladeCenter management modules and are supported).

We are investigating scalable methods for supporting simultaneous and asynchronous execution of multiple instances of agents. We plan to work on providing a more robust error handling mechanism. We also plan to extend Sysman to cover the management of virtual machines as well. Using the interface for managing Linux KVM [7], Xen [17] and VMware [11] virtual machines, Sysman will be used to create and manage virtual machines. This will include the management of storage volumes for these virtual machines as well. Another direction for future is to enhance Sysman to become a distributed management system in which multiple Sysman servers cooperate with each other to provide a higher level of scalability and fault tolerance. We are investigating the use of multicast messages for issuing the same commands to multiple systems too.

In this paper, we presented Sysman, a new infrastructure for system management. Sysman provides a simple yet very powerful interface which makes the automation of system management tasks very easy. Since Sysman is presented to end users as a virtual file system, all UNIX file system commands can be utilized to manage a cluster of possibly heterogeneous servers and other devices. Sysman is designed to be easily extensible. Agents for new devices can be developed and added to the system easily. Furthermore, graphical and web based interfaces can be added on top of Sysman.

Author Biographies

Mohammad Banikazemi is a research staff member at IBM T. J. Watson Research Center. His research interests include computer architecture, storage systems and simplifying server and storage management. He has a Ph.D. in Computer Science from Ohio State University, where he was an IBM Graduate Fellow. He has received an IBM Research Division Award and several IBM Invention Achievement Awards. He is a Senior member of both the IEEE and the ACM.

David Daly is a research staff member at IBM T. J. Watson Research Center. He received a B.S. degree in computer engineering in 1998 from Syracuse University. He received Ph.D. and M.S. degrees in Electrical Engineering in 2005 and 2001 respectively, both from the University of Illinois at Urbana-Champaign. Since joining IBM in 2005, he has worked on computer system performance analysis and modeling, analyzing financial workloads for high performance computing, and simplifying server management.

Bulent Abali received his Ph.D. from the Ohio State University in 1989. He has been a research staff member at IBM Research since then.

Bibliography

[1] *Cluster Systems Management (CSM)*, http://www.ibm.com/systems/clusters/software/csm/index.html .

[2] *Configfs*, http://www.mjmwired.net/kernel/Documentation/filesystems/configfs/ .

[3] *File Systems Using FUSE*, http://fuse.sourceforge.net/wiki/index.php/FileSystems .

[4] *FUSE: File System in Userspace*, http://fuse.sourceforge.net/ .

[5] *Ganglia*, http://ganglia.sourceforge.net/ .

[6] *Introduction to Named Pipes*, http://www.linuxjournal.com/article/2156 .

[7] *Kernel Based Virtual Machine*, http://kvm.qumranet.com/kvmwiki .

[8] *Nagios*, http://www.nagios.org/ .

[9] *Sysmanfs: A Virtual Filesystem for System Management*, https://sourceforge.net/projects/sysmanfs .

[10] *The openMosix Project* http://openmosix.sourceforge.net/ .

[11] *VMware*, http://www.VMware.com .

[12] Amar, L., A. Barak, E. Levy, and M. Okun, "An On-Line Algorithm For Fair-Share Node Allocations in a Cluster," *Proceedings of Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, 2007.

[13] Amar, L., J. Stoesser, A. Barak, and D. Neumann, "Economically Enhanced Mosix for Market-based Scheduling in Grid OS," *Proceedings of Workshop on Economic Models and Algorithms for Grid Systems (EMAGS) 2007, 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, 2007.

[14] Barrett, R., E. Kandogan, P. P. Maglio, E. Haber, L. A. Takayama, and M. Prabaker, "Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices," *Proceedings of ACM Conference on Computer Supported Cooperative Work*, 2004.

[15] Begnum, K., "Managing Large Networks of Virtual Machines," *LISA '06: Proceedings of the 20th Conference on Large Installation System Administration*, p. 16, 2006.

[16] Daly, D., J. H. Choi, J. E. Moreira, and A. P. Waterland, "Base Operating System Provisioning and Bringup for a Commercial Supercomputer," *Proceedings of The Third International Workshop on System Management Techniques, Processes and Services (SMTPS)*, 2007.

[17] Dragovic, B., K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.

[18] Faulkner, R. and R. Gomes, "The Process File System and Process Model in UNIX System V," *USENIX Winter*, pp. 243-252, 1991.

[19] *IBM BladeCenter*, http://www.ibm.com/systems/bladecenter/ .

[20] Massie, M. L., B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, Vol. 30, Num. 7, 2004.

[21] McNett, M., D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: An Extensible Framework for Managing Clusters of Virtual Machines," *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November, 2007.

[22] Minnich, R. and A. Mirtchovski, "Xcpu: A New, 9p-Based Process Management System for Clusters and Grids," *Proceedings of IEEE International Conference on Cluster Computing*, 2006.

[23] Mirtchovski, A. and L. Ionkov, "Kvmfs: Virtual Machine Partitioning for Clusters and Grids," *Proceedings of The Linux Symposium*, 2007.

[24] Pike, R., D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom, "The Use of Name Spaces in Plan 9," *Operating Systems Review*, Vol. 27, Num. 2, 1993.

[25] *RFC 2608, Service Location Protocol (SLP), Version 2*, http://tools.ietf.org/html/rfc2608 .

[26] Sacerdoti, F., M. Katz, M. Massie, and D. Culler, "Wide Area Cluster Monitoring with Ganglia," *Proceedings of IEEE International Conference on Cluster Computing*, 2003.

Devolved Management of Distributed Infrastructures With Quattor

Stephen Childs – Trinity College Dublin, Ireland
Marco Emilio Poleggi – INFN-CNAF, Bologna, Italy
Charles Loomis – Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France
Luis Fernando Muñoz Mejías – Universidad Autónoma de Madrid (UAM), Spain
Michel Jouvin – Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France
Ronald Starink – Nikhef, Amsterdam, The Netherlands
Stijn De Weirdt – Universiteit Gent, Ghent, Belgium
Germán Cancio Meliá – European Organization for Nuclear Research (CERN), Geneva, Switzerland

ABSTRACT

In recent times a new kind of computing system has emerged: a distributed infrastructure composed of multiple physical sites in different administrative domains. This model introduces significant new challenges: common configuration parameters must be shared, local customization must be supported, and policy domains must be respected. We believe that such features can best be implemented by a system that provides a high-level configuration language (allowing structuring and validation of configuration information) and that is modular (allowing for flexible structuring of the overall infrastructure).

The Quattor configuration management toolkit has been designed to meet these requirements. Quattor uses a declarative model where high-level descriptions are translated into configurations and enacted by autonomous components running on the configured machines. Quattor's Pan language provides features for composing complex configuration schemes in a hierarchical manner, for structuring configuration information along node or service lines, and for validating parameters. Finally, the modular architecture of Quattor allows great flexibility in the placement of configuration servers within a distributed infrastructure.

Quattor is being successfully used to manage distributed grid infrastructures in three countries. The suitability of the Pan language is demonstrated by the fact that a comprehensive distribution of configuration templates has been developed as a community effort.

Introduction

Distributed computing paradigms such as grid and cloud computing are changing the face of system configuration management. The traditional computing center made up of hundreds or thousands of computers located at a single site is being reconstituted as a federated system where resources are spread across multiple sites. Each of these collaborating sites, whether independent institutions or departments within a larger organization, requires enough autonomy to implement local policies, and hence the management of the overall infrastructure must be devolved. Nevertheless, a consistent view of the overall system must be provided to resource users. Management strategies for distributed infrastructures have an inherent tension: effective mechanisms for sharing common configuration must be provided without restricting the independence of individual sites.

The high-energy physics community has extremely demanding data processing requirements and has been involved in large-scale computing for decades. It was one of the first communities to see the need to rethink the traditional computing center model, and now concentrates on the creation and maintenance of a global grid infrastructure. An important consequence has been a focus on tools for managing the "fabric" of the grid, namely, the hardware and software installed at sites around the world. Such tools must be scalable to deal with the large installations found at grid sites, flexible to deal with complex services on heterogeneous resources, and modular to deal with the wide range of site structures (including distributed sites). Quattor was designed and implemented to deliver an infrastructure management system that meets these needs.

Grid infrastructures have now moved beyond the prototype stage and one of the biggest challenges currently being faced is the provision of sustainable, managed, and monitored production system. In order to achieve this, a number of grid initiatives have put in place structures for aggregating multiple locations as logical sites. In some cases, this arrangement mainly affects administrative and support structures. In other cases, an integrated configuration management system is put in place to reduce the management load on any one site by allowing individual institutions within a logical site to share common configuration parameters and tools. We believe this is the better approach and that Quattor is an excellent tool for implementing such a system.

Principles

The challenge of structuring and sharing components in a collaborative system is not new; over the years programming language designers have attacked this problem from many angles. While trends change, the basic principles are well understood. Features such as encapsulation, abstraction, modularity, and typing produce clear benefits. We believe that similar principles apply when sharing configuration information across administrative domains.

The Quattor configuration toolkit derives its architecture from LCFG [1], improving it under several aspects. At the core of Quattor is Pan, a high-level, typed language with flexible include mechanisms, a range of data structures, and validation features familiar to modern programmers. Pan allows collaborative administrators to build up a complex set of configuration templates describing service types, hardware components, configuration parameters, users, etc. The use of a high-level language facilitates code reuse in a way that goes beyond cut-and-paste of configuration snippets. (See the ''Pan Language'' section.)

The principles embodied in Quattor are in line with those established within the system administration community [2, 3]. In particular, all managed nodes retrieve their configurations from a configuration server backed by a source-control system (or systems in the case of devolved management). This allows individual nodes to be recreated in the case of hardware failure. Quattor handles both distributed and traditional (single-site) infrastructures (see Table 1).

We consider devolved management to include the following features: consistency over a multi-site infrastructure, multiple management points, and the ability to accommodate the specific needs of constituent sites. There is no single ''correct'' model for a devolved infrastructure, thus great flexibility is needed in the architecture of the configuration system itself. Sometimes a set of highly-autonomous sites wish to collaborate loosely. In this case each site will host a fairly comprehensive set of configuration servers, with common configuration information being retrieved from a shared database and integrated with the local configuration. This is the model used in the Belgian grid infrastructure BEGrid [4]. In a closer collaboration, it may be desirable to centralize the majority of services (configuration database and software package repository), with a bare minimum of services (maybe just node installation) hosted at the individual sites. Grid-Ireland, the Irish grid initiative, is organized along these lines [5].

Distributing the management task can potentially introduce new costs. For example, transmitting configuration information over the WAN introduces latency and security concerns. Quattor allows servers to be placed at appropriate locations in the infrastructure to reduce latency, and the use of standard tools and protocols means that existing security systems (such as a public key infrastructure) can be harnessed to encrypt and authenticate communications.

Applicability

Theory is one thing, but practical implementation brings its own significant challenges. In this paper we use the GRIF [6] deployment, the research grid infrastructure in the Paris region, to illustrate the strengths of Quattor for distributed management. GRIF was formed in 2005 as a collaboration between five sites wishing to present a unified view of their disparate resources. The goal was to amortize the management load of complex grid software by collaborating on the development of shared configuration templates, and to improve service quality by sharing automatically best practices via these templates. The requirements of GRIF, along with those from other infrastructures, have driven the development of a comprehensive set of Quattor templates, known as the Quattor Working Group (QWG) templates [7]. This can be thought of as a ''configuration distribution'' (after the model of a Linux distribution); it gathers together all the potential settings needed to set up basic operating system (OS) services and middleware.

GRIF was built on the foundation of the pre-existing LAL [8] Quattor configuration, which at that time was managing 20 machines. Since the inception of GRIF in 2005, all five sites have added machines to reach the current (2008) number of more than 600 machines spread over six locations, with 100-200 more expected next year. LAL also uses Quattor to manage non-grid servers (now roughly 50 machines) and Linux desktops (25); other GRIF sites are also starting to manage non-grid systems with Quattor.

One of the main challenges that the development of the QWG templates had to address was the shortage of manpower within GRIF. There is little dedicated manpower, and many of the technical team only work part-time on administration. In 2005, only three people had significant Quattor and/or grid experience; Quattor has enabled the incremental growth of a team which today consists of 20 people with good skills. However, due to the support for sharing configuration, only two to three people need to be involved in the development of core templates. Quattor and QWG, despite the somewhat steep learning curve, allow this ''sustainable'' model wherein the operational overhead is minimized allowing people to focus on the services for which they are responsible.

The rest of the paper is organized as follows. The next section describes a distributed management workflow and shows how it can be implemented in Quattor. We flesh out the theory by exploring real deployments of Quattor-managed distributed infrastructure in the ''Real-world distributed management'' section.

Distributed sites often face several management issues, so we present our deployment experiences and then discuss related work, and finally, we outline conclusions and future work.

Devolved Workflow for Distributed Management

We now present a typical workflow for devolved management of a distributed infrastructure and show how it can be implemented using Quattor. For a more in-depth treatment of individual Quattor components, please see a previous work [9].

Figure 1 illustrates the entire ''ecosystem'' of a devolved management infrastructure, showing the typical workflow involved in creating and deploying configurations. To start, administrators create or edit configuration source files, called *templates*, describing the services and nodes in the system. These templates are written in Pan [10, 11], a high-level, declarative configuration language. Administrators then store these templates in a *configuration database*, which may aggregate configuration templates from various sources. The configuration templates are then processed by the Pan compiler which validates their content and compiles them into an XML representation. The resulting XML *profiles* are stored in a *machine profile repository*; there is exactly one profile per managed node. Each managed node retrieves and caches its own profile and autonomously aligns its state with that described in the profile. Corrective actions are performed by specialized agents, or components, which are triggered upon changes in the part of the configuration with which they are registered.

Administrators may or may not be located at the same site as their managed nodes. In the figure, the organization bar.org manages its own machines; in contrast, the management of machines in foo.org is devolved to administrators in baz.org. In fact, Quattor's architecture means that the communication between any stages in the workflow may be carried out across site boundaries.

Configuration Management System

Quattor's configuration management system is composed of a configuration database that stores high-level configuration templates, the Pan compiler that validates templates and translates them to XML profiles, and a machine profile repository that serves the profiles to client nodes. Only the Pan compiler (see the ''Pan Language'' Section) is strictly necessary in a Quattor system; the other two subsystems can be replaced by any service providing similar functionality.

Devolved management in a cross-domain environment requires users to be authenticated and their operations to be authorized. For the configuration database, we chose to adopt X.509 certificates[1] because of the support offered by many standard tools,

[1]Kerberos 5 tickets and encrypted passwords are supported as well.

	Distributed			Single-site			
Metric	BEGrid	Grid-Ireland	GRIF	CERN	CNAF	Nikhef	UAM
Managed machines	260	417	619	8000	800	301	553
Administrators	8	11	25	100	10	4	3
Physical sites	6	18	6	1	1	1	1

Table 1: Quattor deployments.

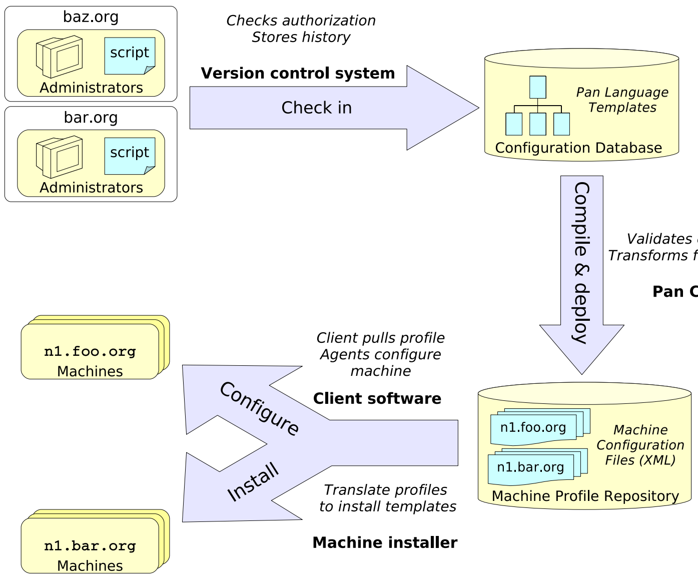


Figure 1: Configuration data workflow in Quattor.

and access control lists (ACLs) because they allow a fine-grained control (an ACL can be attached to each template). When many users interact with the system, conflicts and misconfigurations may arise which require a roll back mechanism; to this purpose, a simple concurrent transaction mechanism, based on standard version control systems, was implemented.

Quattor's modular architecture allows the three configuration management subsystems to be deployed in either a distributed or centralized fashion. In the *distributed* approach, profile compilation (at development stage) is carried out on client systems, templates are then checked in to a suitable database, and finally the deployment is initiated by invoking a separate operation on the server. The *centralized* approach provides strict control of configuration data. The compilation burden is placed onto the central server, and users can only access and modify templates via a dedicated interface.

Since the two paradigms provide essentially the same functionality, the choice between them depends on which fits the management model of an organization better. For instance, the centralized approach fits large computer centers well because of its strictly controlled workflow, whereas multi-site organizations such as GRIF prefer the distributed approach because it allows different parts of the whole configuration set to be handled autonomously. In this paper, we focus on the distributed approach (see the "Real-world Distributed Management" section) as it fits best with the devolved management model we are presenting.

Pan Language

The Pan language compiler sits at the core of the Quattor toolkit. It compiles machine configurations written in the Pan configuration language by system administrators and produces XML files (profiles) that are easily consumed by Quattor clients. The Pan language itself has a simple, declarative syntax that allows simultaneous definition of configuration information and an associated schema. In this section, we focus only on the Pan features that are relevant to devolved management of distributed sites: validation, configuration reuse, and modularization. The original specification of Pan can be found in [10]; a better description of the current features of Pan is available in [11].

Validation. The extensive validation features in the Pan language maximize the probability of finding configuration problems at compile time, minimizing costly cleanups of deployed misconfigurations. Pan enables system administrators to define atomic or compound types with associated validation functions; when a part of the configuration schema is bound to a type, the declared constraints are automatically enforced.

Configuration reuse. Pan allows identification and reuse of configuration information through "structure templates." These identify small, reusable chunks of Pan-level configuration information which can be used whenever an administrator identifies an invariant (or nearly invariant) configuration subtree.

Modularization. With respect to the original design, two new features have been developed to promote modularization and large-scale reuse of configurations: the *namespacing* and *loadpath* mechanisms.

A full site configuration typically consists of a large number of templates organized into directories and subdirectories. The Pan template namespacing mimics (and enforces) this organization much as is done in the Java language. The namespace hierarchy is independent of the configuration schema. The configuration schema is often organized by low-level services such as firewall settings for ports, account generation, log rotation entries, cron entries, and the like. In contrast, the Pan templates are usually organized based on other criteria like high-level services (web server, mail server, etc.) or by responsible person/group.

The namespacing allows various parts of the configuration to be separated and identified. To effectively modularize part of the configuration for reuse, administrators must be able to import the modules easily into a site's configuration and to customize them. Users of the Pan compiler combine a loadpath with the namespacing to achieve this. The compiler uses the loadpath to search multiple root directories for particular, named templates; the first version found on the loadpath is the one that is used by the compiler. This allows modules to be kept in a pristine state while allowing sites to override any particular template.

Further, module developers can also expose global variables to parameterize the module, permitting a system administrator to use a module without having to understand the inner workings of the module's templates.

The "Real-world Distributed Management" section explains the use of the Quattor Working Group (QWG) templates used to configure grid middleware services. The QWG templates use all of the features of Pan to allow distributed sites to share grid middleware expertise.

Automated Installation Management

A key feature for administering large distributed infrastructures is the ability to automatically install machines, possibly from a remote location. To this purpose, Quattor provides a modular framework called the Automated Installation Infrastructure (AII). This framework is responsible for translating the configuration parameters embodied in node profiles into installation instructions suitable for use by standard installation tools. Current AII modules use node profiles to configure DHCP servers, PXE boot and Kickstart-guided installations.

Normally AII is set up with an install server at each site. However, the above mentioned technologies allow the transparent implementation of multi-site installations, by setting up a central server and appropriate relays using standard protocols [12, 13].

Node Configuration Management

In Quattor, managed nodes handle their configuration process autonomously; all actions are initiated locally, once the configuration profile has been retrieved from the repository. Each node has a set of configuration agents (components) that are each registered with a particular part of the configuration schema. For example, the component that manages user accounts is registered with the path /software/components/accounts. A dispatcher program running on the node analyzes the freshly retrieved configuration for changes in the relevant sections, and triggers the appropriate components. Run-time dependencies may be expressed in the node's profile, so that a partial order can be enforced on components' execution. For example, it is important that the user accounts component runs before the file creation component, to ensure that file ownership can be correctly specified.

By design, no control loop is provided for ensuring the correct execution of configuration components. Site administrators typically use standard monitoring systems to detect and respond to configuration failures. Nagios [14] and Lemon [15] are both being used at Quattor sites for this purpose. In fact, Lemon has been developed in tandem with Quattor, and provides sensors to detect failures in Quattor component execution. We discuss integration of external tools further in the ''Integration with External Tools'' section.

While nodes normally update themselves automatically, administrators can configure the system to disable automatic change deployment. This is crucial in a devolved system where the responsibilities for, respectively, modifying and deploying the configuration may be separated. A typical scenario is that top-level administrators manage the shared configuration of multiple remote sites and local managers apply it according to their policies. For instance, software updates might be scheduled at different times.

Software Management

Quattor offers a software package management framework based on the separation of package *repository* and *configuration*. One or more physical repositories can be placed anywhere provided they are accessible via HTTP; the repositories' contents are made known to the configuration system via special Pan templates. Each time the content of a repository changes, the corresponding template must be regenerated. A node's software composition is specified by package lists placed in the configuration templates: package names are checked against the repository templates, and the resulting list is compiled to a machine profile. In a distributed multi-site scenario, the benefit of this separation is clear: package repositories can be placed in strategic locations, typically close to the user sites, even though package lists are stored in a remote location.

The package manager can be configured to act either in a secure mode, where locally-installed packages are automatically removed, or in a flexible mode, where these packages are ignored. The secure mode allows tight control over packages installed on a node, indeed, the package manager will never try to install anything which is not explicitly listed in the node's profile. Rollbacks can be easily performed as the package manager executes operations transactionally, checking that no requested change will result in a dependency conflict. In flexible mode, several versions of the same package can be installed and the package manager can be configured to respect manual installation of packages not listed in the machine profile. This is ideal for devolved management: even when some central policy strictly dictates the basic software composition, local administrators may experiment with customized setups while having the guarantee that their systems can be cleaned up at any time.

Real-World Distributed Management

The Quattor approach to managing distributed infrastructure proved to be effective in the deployed use case detailed in this section. We will focus on the example introduced earlier: the QWG templates for managing a collection of resources spread across multiple institutions as a single grid infrastructure. We show how Quattor provides sufficient support for sharing configuration between sites and for structuring each site according to its own requirements. We illustrate this using examples from the GRIF deployment, which now (2008) comprises more than 600 machines spread over six locations.

Structuring Shared Configuration

A configuration framework which is based on a corpus of shared templates needs to address three main ''structuring'' issues: template distribution, configuration deployment and organization of the infrastructure's configuration. The following sections illustrate how the QWG framework tackles them.

Structured Configuration Distribution

The configuration of grid software (for example, the Globus [16] and gLite [17] middleware) is in itself a challenge for configuration management systems. Grid software includes a wide variety of logical services (worker node, compute server, storage server, etc.), and it is possible for these to be combined on a single physical node. In a distributed infrastructure, certain parameters are common across entire infrastructures (such as the addresses of central grid servers), there are also many configuration parameters specific to each site (such as network settings, local user details, etc.), and local variations in how the services are configured or combined. The challenge in a distributed infrastructure is how to share common configuration while retaining maximum flexibility for local site customizations.

The solution to this problem in our context has been to develop a core distribution of Quattor templates that can be used with minimal customizations by sites. This distribution, known as the *QWG templates*, is the result of a collaborative effort among a set of European grid sites. This distribution can be used "as is" by collaborating sites; all they need to do is configure site-specific parameters (e.g., network addresses). New services or OS distributions need only to be integrated once into the core distribution, and can then be used by all collaborating sites. Additionally, there is substantial scope for customization. The standard templates contain variables that conditionally include local templates with custom settings, providing great flexibility without requiring modification of the core templates. Together, these features reduce the potential for creating incompatible forks and minimize the number of templates maintained at each site.

The QWG distribution has grown dramatically as new services and OS distributions have been added. At the time of writing, a checkout of the QWG templates includes 2805 Pan templates: 1167 for 10 OS distributions, 964 for grid services, 550 for standard components, 115 example templates, and 9 legacy templates. A large proportion of these templates is automatically generated by processing package lists distributed by OS or grid vendors. Even so, a distribution of this size needs to be carefully structured to be manageable; indeed, Pan namespaces are used extensively to tie templates to locations in the directory structure.

Structured Deployment

In order to use the QWG templates, a site must implement a suitable configuration database as described in the "Devolved workflow for distributed management" section. We have developed the Subversion Configuration Database (SCDB) [18] to provide a configuration database suitable for cross-domain use. Authorization is enforced by the version control system (Subversion by default, although CVS has also been successfully used). In this way, standard credentials can be used and a rich set of source control options are available.

In order to share administration load, several projects employ a single Quattor instance to manage a grid infrastructure distributed over multiple sites. Examples are GRIF [6] in Paris, BEGrid [4] in Belgium, and Grid-Ireland (see Table 1 for characteristics). A feature of such management schemes is that access to the configuration must be controlled based on the identity of the user, so that administrators from different sites can safely implement their own modifications without disrupting other sites. This is easy to implement when configuration information is stored in a version control system.

For example, within GRIF, administrators at each site have full control of their own clusters and site templates, but the right to modify standard and GRIF-wide templates is restricted to a set of experts. This ensures that the core contains only well-tested templates. However, this does not prevent local administrators from deploying a modified version of a core template at their own site. To ensure autonomy for local administrators, somebody who does not have the right to modify a "global" template can make a copy in his own cluster or site area and then modify it. When he has tested and validated his changes, he can submit the new template to one of the experts for integration in the standard template set. This approach encourages contributions from all users, yet enforces strict control over the core distribution.

A variety of techniques is used for integrating templates from the core repository with a particular site's (or infrastructure's) own configuration database. Within GRIF and BEGrid, QWG templates are merged into the local repository. Grid-Ireland uses Subversion's "externals" [19] mechanism which allows direct inclusion from an external repository, effectively inserting a pointer to a particular revision of the core templates, causing them to be automatically updated as required. Local templates are then certified against a particular revision of the core templates, which is then used until new features are

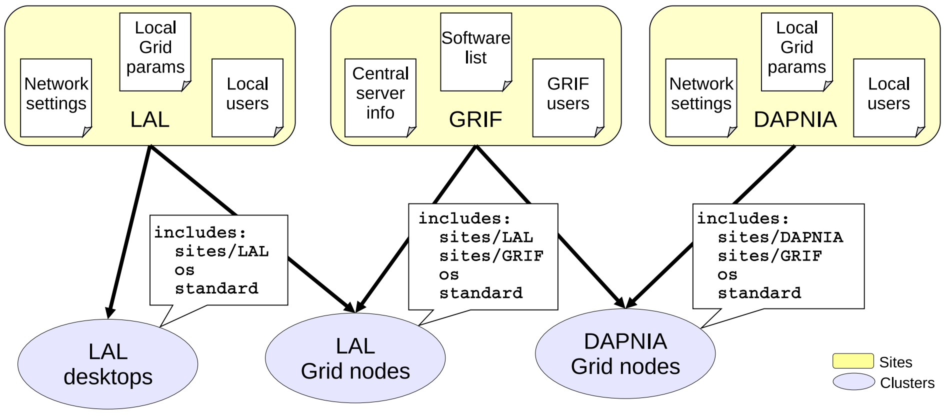


Figure 2: Sites and clusters in QWG.

needed from the core – at this point a new revision of the core templates is certified.

Structured Infrastructure Configuration

The configuration is usually organized using the concepts of *site* and *cluster*. A *cluster* is an arbitrary grouping of machines that share configuration information (for example, "compute nodes" or "grid servers"). A *site* is a logical group that defines a set of configuration elements to be shared by different clusters. For example, in Figure 2, there are three sites. The "LAL" and "DAPNIA" sites contain local configuration data such as network settings and user accounts; the "GRIF" virtual site contains configuration parameters common to all GRIF machines (location of central grid services, common software lists, and user accounts). The LAL and DAPNIA grid clusters include templates from their local site and GRIF, while the LAL desktops cluster only includes the local LAL templates as it is not part of the GRIF infrastructure.

The sites associated with a cluster are specified as an ordered list that defines their precedence in the template search path. For example, for the "LAL Grid nodes" cluster, if the template users.tpl exists in both the LAL and GRIF sites, the version from LAL will be selected as LAL appears first in the list. This technique is used within QWG to create a hierarchical structure; a template in the core distribution can be easily overridden by creating a copy of it in a higher-priority site (e.g., the local site).

Local Customization

Quattor and Pan as used in the QWG templates provide two main methods for customization: loadpaths, which are used for high-level switches between versions and hook variables, which allow the inclusion of custom templates to tweak a specific area of the configuration space.

Loadpaths. Each site is likely to deploy a different mix of base OS types, each of which needs its own package lists and configuration. From time to time, a node (say a web server) will need to be upgraded to a new OS release, while preserving any custom configuration. For example, consider a Xen host machine which needs to install OS-specific RPM packages for the core Xen software; the machine's "object" template includes the template rpms/xen/host:

```
object template xenhost01;
variable LOADPATH = list('os/sl450-x86_64');
include {'rpms/xen/host'};
```

and the template referenced is available for two platforms in two different directories:

```
os/sl450-x86_64/rpms/xen/host.tpl
os/sl510-x86_64/rpms/xen/host.tpl
```

The LOADPATH definition instructs the compiler to prefix its search path with the string os/sl450-x86_64, so that the template found is os/sl450-x86_64/rpms/xen/host. By simply changing the loadpath, the machine profile can be upgraded to use a different version. A similar technique can be applied for version-specific configuration, for example variables that take different values depending on the OS version. In QWG the loadpath for OS templates is set using a simple hash structure that maps machine names to OS version. A configurable default is also provided removing the need to set up a mapping for each machine.

Hook variables. Another approach commonly used in QWG is the use of hook variables that allow sites to integrate their own custom templates. Most of the core templates that configure services or node types contain a conditional include block. The block checks the value of a variable and includes the referenced template if the variable is set.

For example, the template that sets up the basic services for a cluster head node contains the following line:

```
variable CE_CONFIG_SITE ?= null;
# Add local customization to standard
# configuration, if any
include { CE_CONFIG_SITE };
```

In order to customize a local head node, an administrator creates a new template containing the customization (say tcd_torque_config.tpl) and then simply sets the variable CE_CONFIG_SITE to the name of this template. The basic machine type can thus be customized without modifying the core template. Note also the use of ?= in the variable assignment: this is a conditional assignment meaning "assign if a value is not already set." This allows default values to be set that will be used if no local configuration is defined.

A Concrete Example

Figure 3 shows a partial inclusion graph of a worker node foo in the cluster bar at LAL.

The QWG framework defines a set of "machine types" corresponding to typical grid service elements and represented by dedicated templates in the namespace machine-types. The part enclosed in the dashed-line box represents the customization space: these templates are looked up in a cluster-dedicated namespace site/bar/ set externally as an inclusion path to the compiler. A first level of local LAL's site-wide customization is done in machine-types/wn, where a hook variable WN_CONFIG_SITE defines a possible extra template subtree as discussed above: for instance, some of the "local grid parameters," as shown in Figure 2, are set here. A second level of customization for the OS is done in site/cluster_info, where the variable NODE_OS_VERSION_DB defines the template in which the mapping between machine names and platforms is done (site/os/version_db): here foo is mapped to sl450-i386. At this point, all the information for selecting OS-dependent templates is almost completely defined: the last bit is the loadpath which is set in os/version; then config/glite/3.1/base and its cascading templates are searched in the namespace os/sl450-i386. With reference to Figure 2, all the information in this subtree comes from the GRIF site's template set.

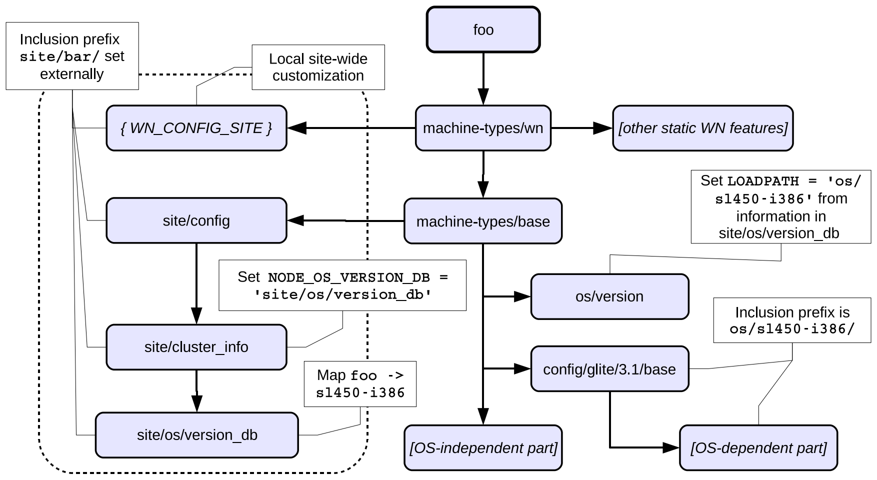
To give an idea of the configuration space size of a worker node, there are 32 platform-dependent templates and 195 platform-independent templates with local site-wide customizations. In addition, there are 25 upstream platform-dependent templates (out of 1208 OS templates) and 138 upstream platform-independent templates (out of 1109 grid and standard templates) without local site-wide customization.

Flexible Deployment Architecture

The use of a high-level language and a source control system facilitate maintainable sharing and customization of configuration information between sites. However, flexibility of deployment architecture is also important for integrating disparate sites.

It is important not to mandate a deployment architecture for sites wishing to collaborate. Local preferences or policies may lead to different implementations at partner sites. Because Quattor is highly modular, each site can choose which elements of their infrastructure to share and which to keep private. Often a similar configuration will be used for all sites within a collaboration, but this is not required. Figure 4 shows a variety of sites participating in a single distributed infrastructure. There is a common configuration database holding core templates and parameters common to all sites, and a shared package repository holding common software. Each site has a set of local machines and an installer that performs their initial installation.

- **Site 1** is an advanced site with a high degree of customization. It hosts its own local configuration database with site-specific templates. These are combined with the core templates from the common database to generate a complete configuration for the site, with machine profiles served locally. This is essentially the model used within BEGrid [4].
- **Site 2** still requires local configuration but to a lesser extent. It does not install custom software locally and so uses the shared package repository exclusively. While the local configuration database is logically distinct, it may actually be hosted on the same server as the shared repository. This is the case within GRIF [6].
- **Site 3** is completely dependent on the shared configuration and is thus centrally managed.



Inclusion prefix site/bar/ set externally
Local site-wide customization
foo
{ WN_CONFIG_SITE }
machine-types/wn
[other static WN features]
site/config
machine-types/base
Set LOADPATH = 'os/sl450-i386' from information in site/os/version_db
Set NODE_OS_VERSION_DB = 'site/os/version_db'
os/version
Inclusion prefix is os/sl450-i386/
site/cluster_info
config/glite/3.1/base
Map foo -> sl450-i386
site/os/version_db
[OS-independent part]
[OS-dependent part]

Figure 3: A partial inclusion graph of a worker node in QWG.

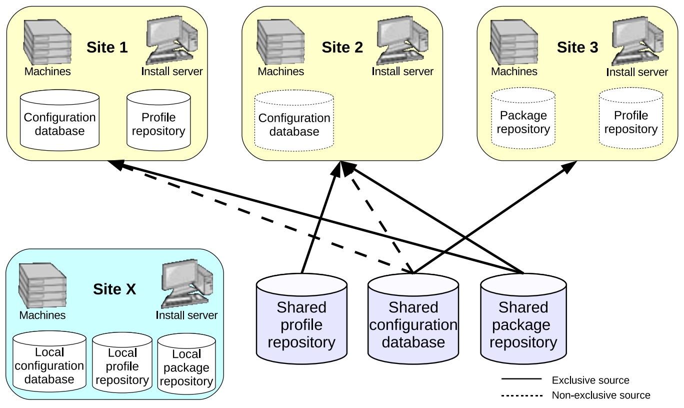


Figure 4: Site configurations in a distributed infrastructure.

> All configuration and software are sourced from the shared database. High-level templates are compiled to low-level profiles at the central site and pushed out to a machine profile repository at the site. A similar process is used for software; packages are mirrored locally, but are treated as part of the same logical repository. This is the model used within Grid-Ireland [5].

For comparison, Site X is a traditional computing center where all configuration and software packages are retrieved from local servers, as happens at CERN's computer center [20].

Experience with Distributed Deployment

The QWG templates have been used to configure production sites since 2005. As such, we have gathered some experience on using Quattor to manage sites, and we present some reflections here.

What Worked Well

Distributed configuration database. The original installation of Quattor at CERN uses a single configuration database which is accessed via a proprietary interface that enables users to check in and compile templates. This approach does not allow offline working as it requires a connection to the central database at all times. For this reason, a Subversion-based configuration database was developed which allows administrators to treat configuration like source code, using all the features of the version-control system directly. Thus configurations can even be developed and tested (at least to confirm that they compile correctly) on an offline laptop, and then deployed when network connectivity is available again.

Complete representation of system state. One of the key benefits of Quattor profiles is that they hold the complete configuration state of a particular system in a structured form. Within a particular machine's configuration, this means that multiple components can access the system configuration (for example, the list of users defined could be accessed both by the ssh component and the accounts component). Machines can also access the configurations of other machines, which is useful for configuring monitoring servers that need to aggregate descriptions of the whole site, and for virtual machines, where host VMs need access to the configuration of their guests. This is done in a controlled way, via Pan constructs that allow to access machine profiles' contents.

Namespaces and loadpaths. These two features (new since the earlier Pan paper) have proved to be extremely useful in managing the large set of templates now in use. Namespaces allow hierarchical naming schemes to be enforced, resulting in a clear structure for the overall distribution (for example, a component schema definition template (schema.tpl) with the name components/accounts/schema **must** be located in a directory whose path ends in components/accounts.

Loadpaths have proved useful in keeping machine template code OS-agnostic as detailed earlier in the paper. They allow large portions of a machine or service configuration to be easily switched to a different OS or middleware version (the template does not usually have to be changed at all, as the change is made in an external file). They have also been used to implement "stage-based deployment." In this approach, templates are classified as "development" or "production," and stored under different paths accordingly. Again, the switch from development to production can be quickly affected by changing the value of the loadpath variable. Furthermore, loadpaths are used for installing and configuring different versions of the same component.

What Does Not Yet Work Quite So Well

Here we list some ongoing limitations that impact on the day-to-day task of managing sites, together with our current thoughts on resolving them.

Software dependency management. Quattor's software package manager is declarative: administrators define the exact list of packages to be installed on a node, and the package manager is normally run in an enforcing mode so that no other packages are permitted to be installed locally. This behavior is different from other commonly-used package managers such as YUM and APT which automatically pull in packages to satisfy the dependencies of a newly-installed or updated package.

The advantage of this approach is that administrators have strict control over software on managed nodes. The disadvantage is that package dependencies must be comprehensively defined at configuration time rather than allowing the package manager to resolve them automatically. If a package list is deployed which contains unsatisfied dependencies, then the whole package manager transaction will fail. As many other components depend on the package manager having run successfully, the result is often that the new configuration does not take effect. As this failure mode was only detected after deployment, it proved extremely frustrating (becoming known in the community as "RPM dependency hell").

This is a difficult problem to solve at the time of profile creation because package dependency information is contained in the packages themselves which are typically stored in a central repository. Rather than implementing our own dependency checker, we decided to leverage the YUM package manager, whose functions are easily accessible from Python code. We have written a Python program that processes a given Quattor profile, extracts details of software repositories used, generates YUM repository descriptors to represent them, and then invokes YUM code to indicate whether there are missing dependencies in the RPM set. This approach relies on YUM metadata being kept up to date on the package repository server, but this can easily be automated.

Dependencies on external tools. One of the goals of creating a shared configuration distribution is to reduce the learning curve needed to establish a new Quattor installation. Quattor has been designed to make as much use of standard services as it can. For example, network installation in Quattor uses DHCP, PXE, and Kickstart, as well as relying on correct DNS configuration. In principle this is a good thing as it makes Quattor easier to integrate with existing infrastructures. However, when new, inexperienced people want to join a collaboration with a new site, they must master all of these services (at least to the point where they can debug problems) before they can install a single box!

Fine-grained authorization. As configuration management becomes more widely distributed through a larger number of people, authorization and control become extremely important in maintaining the quality of the deployed machine configurations. A critical issue in distributed management is that of fine-grained access control to restrict users from interfering with configuration that lies outside their sphere of control. To date, this has been handled by enforcing authorization for templates in the configuration database. For example, administrators of a particular cluster might be restricted to editing the templates for their own cluster machines in an effort to prevent them from modifying the configuration of core services. However, this is not a sufficient solution, as any template can modify any part of the configuration namespace. A solution is needed that ties authorization at the template level (enforced by the configuration database) to authorization at the level of portions of the profile namespace (this must be enforced by the compiler).

Structure templates (see the "Pan Language" section) provide a partial solution for this. An administrator allowed to edit such a template can modify only a small and well defined sub-tree of the configuration hierarchy. However, a structure template can read the entire hierarchy, and this still has to be restricted in some way.

Debugging a complex configuration. The trade-off for having so much flexibility is an increased difficulty in debugging a complex configuration instance. Indeed, the high-level Pan representation is based on dynamic includes which are resolved at compilation time, so that, "navigating" the templates becomes almost impossible. This has increasingly generated frustration among QWG users, limiting the possibility of a closer than just operational approach for people who wish to better understand and contribute to Quattor developments. As a response, we are developing some facilities for visualizing and browsing a site's configuration via different graphic formats.

Lessons Learned

Here we describe the features which have proved to be crucial in sharing configuration between sites.

Stability of the core configuration. One of the key problems for early adopters of the QWG template distribution was the instability of configuration mechanisms. The developers were still working out the best way to represent the complex configuration, and so the underlying data structures frequently changed. This caused great frustration for those using the configuration as local templates had to be continually adapted. As a result of this experience, backward-compatibility is now a key goal for the QWG templates. In many cases, this is achieved by hiding the internal details of the data structures behind simpler functions. For example, the configuration of Xen guests is performed by setting various parameters (e.g., the list of guests on a particular machine) and then including the configure_xen_guests template which fills in the relevant part of the profile. The internal representation may change, but the interface for users remains the same.

Low-effort mechanisms for applying updates. Quattor is principally used to configure grid systems that rely heavily on external distributions both for basic OS functionality and grid middleware. Updates to both are regularly released and must be deployed to ensure that systems remain secure. The general approach taken in QWG is to automatically include a template that "upgrades" any packages selected to the latest versions from the OS or middleware's updates. The update list is automatically generated from the OS or middleware updates directory.

Integration With External Tools

Implementing a distributed infrastructure requires peaceful coexistence with systems already in place at various sites. Collaborating sites often have existing commitments to deployed tools for monitoring, management, or reporting. A key requirement for a distributed configuration management system is the ability to integrate smoothly with such local systems. Because configuration information is stored in a structured fashion, it can easily be reused to generate configuration files for new external tools.

This topic is a major focus for the expanding Quattor community as new tools are encountered. In this section we describe some existing integration work that links systems for monitoring, virtualization, and user desktop management into Quattor infrastructures.

Monitoring. Configuration of monitoring tools such as Nagios [14] and Lemon [15] often requires the administrator to define lists of machines with their associated services. This information is needed to determine how to group machines together for reporting and which sensors to read on a given machine.

With Quattor, this information is already available, as configurations are structured in terms of service and node types. Hence the configuration lists needed for monitoring tools can be automatically derived from the configuration database. It has proved straightforward to implement Pan functions that extract information about all existing nodes and provide it to the Nagios templates. These templates can then generate complex, fine-grained policies, for instance *raise an alarm if the load on a single-CPU node goes above three, but hold that same alarm on four-CPU nodes until the load reaches 8*, without explicitly instructing Nagios about how many processors each node has. Also, node-specific checks can be specified, and hosts not listed in the configuration database can be added to the Nagios configuration. This is useful when monitoring external services, such as routers.

Virtualization. A major strength of a Quattor configuration database is the ability to reference configuration parameters from other services or even machines. This has proved particularly useful in the configuration of virtualization, where a number of guest nodes are hosted as virtual machines (VMs) on a host. The configuration of a VM has two dimensions: the configuration of the node itself (its network settings, users, services, etc.) and the configuration needed on the host to run the VM (location of file system storage, virtual MAC address, boot mechanism, etc.). The node configuration resides in the profile as normal, and much of the "external" configuration resides in the machine's hardware description template which lists disk sizes, RAM size, and network cards. The host can thus reference this information and use it to generate the VM configuration files.

In broader terms, Quattor supports virtual machines using the concept of an *enclosure*. An enclosure is an entity composed of a parent and one or many children. The parent is always a Quattor representation of a physical machine whereas a child can represent either a physical or virtual machine. This allows modeling of real hardware enclosures as well as virtual machines.

Quattor-based virtualization management using Xen has already been deployed in production environments [21] and OpenVZ virtualization has also been implemented. A variety of methods for instantiating VM file systems has been used, ranging from full PXE-based automatic installation to bootstrapping from pre-built images. In all cases, AII is used to guide the installation and bootstrap procedure for virtual machines.

User desktops. Once administrators see the benefit of having grid clusters and services under Quattor management, they often look around to see whether other parts of their infrastructure can also benefit. At UAM, Quattor is used to manage Linux installations on about forty user desktops that also have a Windows partition. Typically, desktop users have little knowledge of system administration, and often tend to deviate from security policies, for instance, by installing unauthorized software.

Using Quattor allows administrators to keep all machines in compliance with site's policies. Quattor's management tools can be configured to automatically remove any unauthorized software and user accounts. Other security measures such as firewall rules or controlled privilege escalation (i.e., sudo) can also be configured by the administrators without imposing additional burdens on users.

AII can detect and preserve existing file systems on the disks, and so Windows installations can live side by side with Quattor-managed Linux installations. This is a specific example of a general principle in Quattor, where parts of the machine's configuration can effectively be ignored in order to delegate their management to other tools. This flexibility enables system administrators to "start small," initially just managing a small set of critical services. By gradually extending Quattor management, they can consolidate configuration information, making the most of their investment in Quattor.

Related Work

Fabric management systems abound in both the open source and the commercial sector. Some systems like Oscar [22] and Rocks Clusters [23] use pre-built system images, creating new machines by installing and customizing a pre-built file system. Active Directory [24] provides a hierarchical schema for managing network resources, but it does not easily allow handling arbitrary (i.e., non-Microsoft) services and SW packages. Other systems define declarative languages [3] to facilitate the configurations of machines within the fabric. For a complete survey, see the literature [25, 26]. We believe that systems using high-level declarative languages can provide the flexibility needed to implement distributed management infrastructures. Here, we compare the language features in four systems – LCFG [1], Cfengine [27], Puppet [28], and PoDIM [29] – to those in Pan.

LCFG relies on configuration source files expressed in a high-level declarative language which are transformed into low-level, machine-readable XML profiles, similar to Quattor. However, LCFG's configuration language, based on the C pre-processor, is limited when compared to Pan. It lacks user-defined types, user-defined functions, and validation constructs for expressing constraints (although see PoDIM discussion below). It doesn't provide any equivalent to Pan's namespaces and loadpath, whose importance we have demonstrated for devolved management of remotely distributed clusters based on different architectures. A number of the authors have experience with using LCFG to manage grid sites. The main limitations we found in practice were the lack of an overall configuration schema, which made it difficult to share information between components, and the lack of easy-to-use programming-language features such as hash data structures and iterators.

Cfengine is a policy-based configuration management system in which each managed host belongs to one or more *classes* for which some *policies* apply.

Since a policy specifies the actions requested to align a part of the system with the desired state, the configuration description is partly *procedural* [3]. This makes it difficult to reason about the relations of different configuration parts; in fact, there is no validation support. Although Cfengine's language provides functions for manipulating parts of the managed system, such as symbolic links, it lacks more refined constructs like type definitions and inclusion statements. This paradigm does not easily allow hierarchical schemes, limiting the possibilities for devolved management over distributed sites.

Puppet uses a high-level declarative language with a feature set similar to that of Pan's. In the Puppet language one can define *resources* with associated attributes. The language permits the administrator to define values as well as providing defaults. In addition, *classes* can be defined to group resources into high-level units. Further, *modules* can be defined and shared with others. The real difference with Pan is the validation. Puppet validates the schema on the server, but only validates the attribute values once the configuration is instantiated on the client. This provides late notification of problems and limits the possibilities of cross-machine validation.

PoDIM is a recently proposed language aiming, like Pan, to provide high-level configuration management [29]. Compilers for both languages operate in the same manner, converting source configuration files into a series of XML configuration files, one for each managed node. Pan uses a declarative syntax and PoDIM uses a rule-based one. One distinctive feature of PoDIM is its ability to define cross-machine constraints via the rules declared by the system administrator. Pan provides similar features via validation functions that can verify consistent configurations between machines. The primary difference is how inconsistent configurations are resolved. PoDIM solves for alternate configurations consistent with the declared rules; Pan requires that a system administrator change the configuration to resolve the identified problem.

Other important features of high-level configuration languages are the ability to define a configuration schema, mechanisms for propagating common configuration parameters between machines, and multiple inheritance of attributes and constraints. Both languages have facilities for all of them. PoDIM defines a schema through class attributes, rules, and invariants; Pan, as shown above, uses an extended typing system. In Pan, a managed object may only read information from another object's configuration. This is accomplished through use of the `value()` function and is critical for cross-object validation. PoDIM uses commands to allow managed objects to set attributes within other objects. PoDIM naturally allows multiple inheritance through the underlying Eiffel implementation. With Pan, any path can have multiple types bound to it, thereby allowing multiple inheritance of validation constraints.

PoDIM has an internal mechanism to authorize (or deny) certain changes to a configuration based on identity, an important feature in environments where many people maintain a fabric's configuration. Pan currently has no equivalent functionality. The Quattor toolkit instead controls access to particular source configuration files; this avoids accidental modifications but cannot prevent malicious changes.

Recent collaborative work between the LCFG and PoDIM teams [30], has shown that integration of the two is possible. It also identified a couple issues (slow performance and configuration oscillation) that will need to be eliminated before PoDIM can be used in production. The paper also indicates that a unified language combining the features of LCFG and PoDIM is desired; Pan to a large extent does combine them. Moreover, production use of Pan has been shown to be scalable and manageable, handling up to 8K machines efficiently in the deployment at CERN's computing center.

Conclusions and Future Work

With the adoption of grid computing, it has become increasingly common for different institutions to group their resources and present them as a single logical site. The consequence is a new type of fabric – a co-managed, distributed infrastructure composed of multiple physical sites in different administrative domains. Quattor meets the main requirements for the management of such a system: sharing of common configuration data, possibility of local site customizations, use of standard secure communication protocols, and flexibility.

Quattor's effectiveness mostly comes from the use of Pan, a declarative high-level configuration language which allows the definition of hierarchical configuration schemes using a simple syntax. The namespacing and loadpath features permit large-scale reuse of a configuration's parts and foster the collaborative deployment and management practices which enable the use case described in this paper. The most interesting result of this collaboration is a complete configuration framework, the QWG templates, intended for the configuration of grid services.

Having more actors on the scene requires a tighter control. Pan supports validation, that is, definition and checking of constraints before deployment. Also, appropriate means for authenticating and authorizing users and machine profile transfers are provided by Quattor. The result is an effective devolved management mechanism, with a reduced probability of service disruption due to misconfigurations or mischievous interventions.

Quattor's modularity and use of standard protocols make it attractive in a wide spectrum of different site configurations. The elements of the Quattor toolkit – the configuration databases, tools for software installation management, tools for package repository management, and automated installation facilities – can be used, ignored, or replaced to meet the specific needs of a site, while peacefully coexisting with other management tools. The proof of this flexibility is the variety of use cases described in this paper: BEGrid relying mainly on local customizations for the configuration, GRIF which uses a local configuration repository and a shared package repository, and Grid-Ireland which depends completely on shared, centrally managed, repositories.

The Quattor toolkit is a mature solution used by a large number of diverse sites. Mature, however, does not mean static: the Quattor toolkit continues to evolve incrementally in response to the needs of the Quattor community, which now includes industrial as well as academic users. While Quattor usage is still concentrated within the academic grid community, the model we propose is more widely applicable. For example, large multi-national corporations already run their own grid-like infrastructures spread across geographical locations. Policies at individual sites may differ due to local historical or legal constraints, resulting in the same requirements for sharing and specialization we have already described. Mergers and acquisitions may result in an ever-changing set of policy domains and local tools that must all be integrated into a coherent whole. These are similar problems to those faced by distributed grid sites; similar solutions could be applied.

Quattor also functions well as a general fabric management system and is being used successfully in other, more traditional scenarios, such as that of a large centralized computer center like at Nikhef, CNAF, and CERN. In fact, many of those who started using Quattor to manage their grid infrastructures are now putting other aspects of their operations (mail servers, desktops, etc.) under Quattor control.

Quattor is a complex tool with a steep learning curve: although the community provides an active support, we need to improve our knowledge base, which entails providing clear use cases and possibly out-of-the-box solutions. We are also actively working on mitigating the difficulties encountered by new users via tools for visualizing a site's configuration. Another area for further development is the integration with tools for handling migration of virtual machines: this is somewhat conflicting with the base line dictating that a node should look as it is declared to be, since dynamically changing the location of a virtual machine necessarily modifies the node's configuration. Moreover, in response to increasingly rising security concerns, we are discussing how to extend Pan to support an authorization mechanism that directly protects parts of the configuration schema (not just templates). On the same line, we are working on integrating Quattor with SELinux: we need to define minimum contexts for the core services and confine them accordingly, as well as to manage target nodes' configuration via an NCM component.

Acknowledgements

The first version of Quattor was a product of the European DataGrid project and subsequent versions have been used to manage some grid resources of the EGEE series of projects. The authors gratefully acknowledge the European Commission's support of all of those projects. We would like to thank Kent Skaar and Narayan Desai for shepherding and Paul Anderson for encouraging the writing of this paper.

Author Biographies

Stephen Childs is an Associate Research Lecturer and Research Fellow at Trinity College Dublin. He is the Deputy Grid Manager for Grid-Ireland, and is responsible for fabric management and virtualisation. He is an active in grid operations and parallel jobs support within the EGEE project and a contributor to Quattor. He received a Ph.D. in Computer Science from the Cambridge University Computer Laboratory in 2002, and a B.Eng. in Computer Engineering from the University of Limerick in 1997.

Marco Emilio Poleggi received his M.Sc. in Electronic Engineering from University of Rome "La Sapienza" and his Ph.D. in Computer Engineering from the University of Rome "Tor Vergata." During his studies, he focused on cache cooperation mechanisms for cluster-based web servers. He worked at CERN as a research fellow from 2005 to 2007, developing Quattor, for which he was also appointed release manager. He currently works at INFN-CNAF – the Italian National Center for Telematics and Informatics – where he manages the service configuration for the storage group. Marco Emilio is a member of the IEEE Computer Society.

Charles Loomis is currently a research engineer with CNRS in France and a founding partner of Six^2 Sàrl in Geneva, Switzerland. In 1992, he received his doctorate from Duke University in high-energy physics and subsequently worked as a researcher in major experiments on both sides of the Atlantic. His participation in grid technology projects started in 2001 with the European DataGrid project and continues to this day, within the EGEE series of projects. He initially played a major role in integrating, testing, and deploying the grid middleware. Now, he is responsible for the "application" activity within EGEE, working with end-users to ensure that the EGEE middleware and infrastructure meets their needs. Over this period, he has been a major contributor to the Quattor toolkit and is responsible for the Pan language compiler.

Luis Fernando Muñoz Mejías obtained his degree on Computer Science and Engineering at UAM (Autonomous University of Madrid, Spain) in 2006. He currently works as system administrator at UAM's site of the Spanish LCG cloud. As a member of the Quattor community, Luis Fernando is responsible of the installation framework.

Michel Jouvin has been a system administrator for 25 years. He joined CNRS/LAL in 1992 and had responsibilities in many different areas; he joined the grid effort in the early 2000s. He became technical manager of the GRIF grid site when it was created in 2005. Michel has been involved with Quattor since 2004. He has been a contributor to various Quattor components and had a major role in re-engineering the first generation of QWG templates into a generic framework. He initiated the collaborative effort for documenting the Quattor tools, in particular the QWG wiki. He also participates in several EGEE/LCG bodies involved in grid operations.

Ronald Starink works as Research Associate at the National Institute for Subatomic Physics (Nikhef) in Amsterdam. Since 2005 he is responsible for grid system administration at Nikhef, which includes operations, storage systems and fabric management. He participates in the operations activity of the EGEE project and contributes to the Quattor community as maintainer of various components. Ronald received his Ph.D. in nuclear physics from the Free University Amsterdam in 1999. Subsequently he worked as a software engineer in the telecommunications industry and in business automation. In 2004 he joined Nikhef as software engineer, working on the grid project Virtual Laboratory for e-Science.

Stijn De Weirdt obtained his M.Sc. in physics at University of Ghent, Belgium in 2000. He started to work in November 2004 as system administrator at Free University of Brussels to setup a grid site for LCG (LHC Computing Grid) and BEGrid (Belgian research grid). For BEGrid, he integrated Quattor in the management of the distributed infrastructure. Since July 2008, he started to work at the University of Ghent to coordinate several scientific computing projects.

German Cancio received his M.Sc. after studying Computer Science at the UPM (Polytechnic University of Madrid, Spain) and the University of Savoie, France. After being a Research Assistant at the Artificial Intelligence Department of the UPM, German moved on to CERN in 1998 working on various activities related to large systems administration. He was the Fabric Management Architect of the European DataGrid project (2001-2004) and responsible for the development of Quattor until 2007. Since 2008, German is leading a team charged with the development of grid and mass storage management tools for CERN's Computing Grid. German is a member of USENIX, ACM and IEEE Computer Society.

Bibliography

[1] Anderson, Paul and Alastair Scobie, "LCFG – The Next Generation," *UK UNIX User Group Winter Conference*. UKUUG, 2002.

[2] Traugott, Steve and Joel Huddleston, "Bootstrapping an Infrastructure," *Proceedings of the 12th Large Installations Systems Administration (LISA '98) Conference*, 1998.

[3] Anderson, Paul, "A Declarative Approach to the Specification of Large-Scale System Configurations," 2001, http://www.dcs.ed.ac.uk/home/paul/publications/conflang.pdf .

[4] *The Belgian Grid for Research*, http://www.begrid.be/ .

[5] Coghlan, B. A., J. Walsh, and D. O'Callaghan, "Grid-Ireland Deployment Architecture," in Peter M. A. Sloot, et al., editors, *Advances in Grid Computing – EGC 2005*, LNCS3470, Springer, Amsterdam, The Netherlands, February, 2005.

[6] *GRIF – Grille de Recherche d'Ile de France*, http://www.grif.fr/ .

[7] *LCG Quattor Working Group (QWG templates)*, https://trac.lal.in2p3.fr/LCGQWG .

[8] *LAL – Laboratoire de l'Accélérateur Linéaire*, http://www.lal.in2p3.fr/ .

[9] García Leiva, R., et al., "Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics," *Journal of Grid Computing*, Vol. 2, Num. 4, 2004.

[10] Cons, L. and P. Poznański, "Pan: A High-Level Configuration Language," *Proceedings of the 16th Large Installations Systems Administration (LISA '02) Conference*, 2002.

[11] *Pan Configuration Language Reference*, https://trac.lal.in2p3.fr/LCGQWG/wiki/Doc/panc .

[12] Droms, R., *RFC 2131 – Dynamic Host Configuration Protocol*, Technical report, Network Working Group, 1997.

[13] Patel, B., B. Aboba, S. Kelly, and V. Gupta, *RFC 3456 – Dynamic Host Configuration Protocol (DHCPv4) Configuration of IPsec Tunnel Mode*, Technical report, Network Working Group, 2003.

[14] *Nagios, An Open Source Host, Service, and Network Monitoring Program*, http://www.nagios.org/ .

[15] *LEMON – LHC Era Monitoring*, http://lemon.web.cern.ch/lemon/index.shtml .

[16] *Globus Toolkit, An Open Source Software Toolkit Used For Building Grids*, http://www.globus.org/toolkit/ .

[17] *gLite, An Open Source Grid Middleware Developed by the Egee Project*, http://glite.org/ .

[18] Jouvin, Michel, *Subversion Based CDB*, https://trac.lal.in2p3.fr/LCGQWG/wiki/Doc/SCDB .

[19] Subversion, *Externals Definitions*, 2008, http://svnbook.red-bean.com/en/1.4/svn.advanced.externals.html .

[20] Meliá, G. Cancio, et al., "Current Status of Fabric Management at CERN," *Computing in High Energy and Nuclear Physics (CHEP) Conference*, Interlaken, Switzerland, September 2004.

[21] Childs, Stephen and B. A. Coghlan, "Integrating Xen with the Quattor Fabric Management System," Michael Alexander and Stephen Childs, editors, *Euro-Par 2007 Workshops (VHPC '07)*, Num. 4854 in LNCS, pp. 214-223, 2007.

[22] *OSCAR – Open Source Cluster Application Resources*, http://oscar.openclustergroup.org .

[23] *Rocks Clusters*, 2008, http://www.rocksclusters.org/ .

[24] *Windows Server 2003 Active Directory*, 2008, http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.mspx .

[25] Poznański, Piotr, *Framework for Managing Grid-enabled Large Scale Computing Fabrics*, Ph.D thesis, AGH University of Science and Technology Krakow, Poland – Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, 2005.

[26] Delaet, Thomas and Wouter Joosen, *Survey of Configuration Management Tools* Technical Report, Katholieke Universiteit Leuven, Department of Computer Science, April, 2007.

[27] Burgess, M., "Cfengine: A Site Configuration Engine," *USENIX Computing systems*, Vol. 8, Num. 3, 1995.

[28] *Puppet*, http://reductivelabs.com/trac/puppet .

[29] Delaet, Thomas and Wouter Joosen, "PoDIM: A Language for High-level Configuration Management," *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*, USENIX Association, 2007.

[30] Delaet, Thomas, Paul Anderson, and Wouter Joosen, "Managing Real-World System Configurations with Constraints," Technical Report, Katholieke Universiteit Leuven, Department of Computer Science, 2007.

Authorisation and Delegation in the Machination Configuration System

Colin Higgs – University of Edinburgh

ABSTRACT

Experience with a crudely delegated user interface to our internally developed configuration management system convinced us that delegated access to configuration systems was worth pursuing properly. This paper outlines our approach to authorising access both to individual aspects of configurations and to collections of configurations. We advocate the use of authorisation of some kind on configuration changes and we believe that the system of authorising primitive manipulations of a configuration representation outlined herein could be accommodated by a number of existing configuration systems. The authorisation system described is still experimental and we regret that real world experience of the system in use with end users is not yet available.

Introduction

There are a number of configuration management systems for computers [12, 13, 15, 6, 7, 14]. Of those, all those known to us assume that use of the configuration management system will be restricted to those people who would have been system administrators on the managed collection of computers in the absence of the configuration management system. One can usually use file-system permissions or other mechanisms (inherited ACLs operating in a similar fashion to file-system permissions in Active Directory [10] or database access controls in SMS) to allow or deny access to the configuration representations of computers, but those mechanisms are difficult to use in a structured manner, do not allow delegated control of sub-parts of machine configuration information and, with the exception of Active Directory's ACLs, are not at all designed with configuration delegation in mind.

There are a number of reasons it is desirable to delegate configuration management as much as possible:

- **Avoiding extra work and bureaucratic delay.** If a system does not delegate some control to end users, those who need something about their system changed (for example a new application added) cannot realise that change without speaking to an operator of the configuration management system. This required interaction between two people creates extra work for the operators (and probably also for the end users). Also, a certain amount of delay between request and action is inevitable, which is undesirable to the end user and usually detrimental to the operation of the business.
  The version of our tool currently in service, which does not include the authorisation features described in this paper, nonetheless allows end users to perform some basic configuration tasks via a graphical interface – namely manipulating packages and printers. From a computer population of 300 and a user population of 200, there are around 400 delegated configuration changes per month. That's 20 change requests per working day that would otherwise need an IT team member to enact. We would expect the rate of change requests to increase slightly if access was delegated to more configuration aspects.
- **End user satisfaction.** End users *like* to feel in control of their computers. A system that gives them as much control as can possibly be given is more likely to be popular.
- **Separation of expertise.** Large organisations – the ones who are most likely to require configuration management tools – have larger teams of administrators. Responsibility may be distributed among these administrators in different ways – some may be responsible for servers, with responsibilities split by server or service, others may be responsible for groups of clients, split by location. Still others may have a responsibility that covers a domain of expertise – for example, networking. It ought to be possible to support the various ways that the organisation might choose to delegate responsibility.

Of course, there are also extra complications when many people can edit configuration information.

- **Conflicts in intent.** When many people from different parts of an organisation can contribute configuration instructions, any given computer might be given *conflicting* instructions from multiple sources. There is no guarantee that those contributing to the configuration of the computer even agree on what it should do. The configuration management system must have a method of resolving such conflicts.
- **Accidental breakage.** End users are not usually, and should not have to be, experts in configuring computers – individually or en mass. This lack of expertise could easily result in

breakage, either of their own computer or further afield unless the system guards against it. An example of this would be trying to make two mutually exclusive configuration changes, or making only one change of a dependent pair.

- **Malicious intent.** If an interface into the configuration management system is provided to everyone, then either the interface or the system itself must guard against input with malicious intent.

Of the systems mentioned above, only Active Directory with its LDAP based hierarchy explicitly allows for delegation and sub-delegation of computer configuration on collections of computers. Active Directory's tree structure suffers from the deficiencies inherent in single inheritance trees outlined later, however. These have a negative impact on the way that configuration information may be organised and composed, and on the way that configuration tasks may be delegated. None of those systems allow delegation of access to *aspects* [2, 3] of the configuration – if one has access to the object representing the computer, one can change anything the configuration management system is capable of controlling.

In this paper, a method for organising and delegating access to configuration information is discussed. These ideas have been implemented in an (as yet) experimental extension to our existing configuration management system, "Machination" [11].

First, strategic goals and desired features and properties of the system are discussed. The paper then covers two main authorisation topics: how to represent and manipulate configuration information such that control over aspects can be authorised; and how to organise configuration information when dealing with many configuring entities and configurable objects.

Goals

With the above justification in mind, the following high level goals were set and used as guidelines to extend our existing configuration management system.

1. **Maximise delegation.** Use delegation to help improve acceptance of the configuration management system and to streamline the configuration change process. Any delegation so introduced should also have authorisation and access controls sufficient to satisfy management that the system will not be misused, where the meaning of "misused" is defined by management.
2. **Structure with flexibility.** The system should be compatible with with the way people and things are organised now, or most naturally – not the other way around. This requires a flexible structure with which to organise both configuration and authorisation information. Delegation should be possible to individual end-users, to experts based on their expertise, over computers based on their organisational affiliation or their purpose or their physical locations, along with a number of other criteria.

At a more detailed level, the following requirements relevant to authorisation were set for the system:

1. **Ability to authorise access to configuration aspects individually.** This is required to achieve the strategic goal of Maximising Delegation, allowing delegated access to people who could not be granted access to the whole configuration. This feature should allow one to guard against malicious intent targeted at an the configuration representation of an individual configurable object.
2. **Ability to authorise access to collections of some kind.** To effectively manage large numbers of configurable objects without large amounts of effort, the configuration systems mentioned above all have facilities to collect objects in some way before applying configuration instructions. The authorisation layer should be capable of applying to the same kinds of collections.
3. **Inheritance and/or aggregation.** It was decided early on that the configuration system should have an inheritance or aggregation structure for configuration instructions. The structure chosen is described later. Although this feature is more targeted towards the organisation of configuration instructions, it has strong implications for the way authorisation should work.
4. **Merging and conflict resolution mechanism.** The organisational structure mentioned above requires a mechanism for merging configuration information from multiple potentially conflicting sources. This mechanism has consequences for authorisation and delegation, as described later.
5. **User interface.** There is no use in giving people permission to do something if they lack the means to do it. Delegating configuration to end users requires a user interface that is suitable for end users to use.
6. **Dependency mechanism.** This is required to alleviate the accidental breakage problem described above. Machination has such a dependency mechanism, but it is not discussed further since it is only peripherally related to authorisation.

Authorising Access to Configuration Aspects

The ability to authorise access to configuration aspects individually requirement, set out above, requires the system to authorise access to individual configuration aspects. These aspects are elements of configuration information which are logically connected in some way – packages, web server configuration, network settings or the like. To facilitate this, a representation for configuration information was sought with the following properties:

- The rules for constructing representations must allow representations that are capable of representing all required configurations!
- One should be able to collect related configuration elements together so that access to related elements can be authorised and delegated sensibly.
- One should be able to utilise a finite, and preferably small, number of primitive instructions to manipulate the representation. Authorisation rules will apply to these primitives, so each primitive should have a limited scope so as to avoid the potential for circumnavigating said authorisation.

In the rest of this section, the particular XML representation used in Machination and the primitives used to manipulate it are described.

The XML Representation

In our case the representation chosen was based on XML with the following restrictions:

1. Mixed content elements are not allowed. Every element should either contain child elements or text – not both. It is permissible for elements to contain nothing (normally carrying their information in their attributes).
2. Every element whose tag *could* appear multiple times within a given parent *must* be given an id attribute, which is *unique* amongst that element's siblings with the same tag.
3. The id attribute must only be used for the purpose outlined above.

Rules 1 and 2 together guarantee that every element in the representation can be *uniquely* referenced by an xpath of the form:

```
/tag/arrayTag[@id='id1']/tag/...
```

This is important for the configuration manipulating instructions described in the next section. In fact, since the id attribute is the only one required to address elements, in many cases xpaths are abbreviated to the form:

```
/tag/arrayTag[id1]/tag/...
```

Rule 3 is stated for completeness, though it is considered unlikely that anyone would wish to use the id attribute for anything else.

Assuming a mechanism exists for translating XML to real configuration (and it does exist) the problem of configuration management has been reduced to producing and distributing valid XML for that mechanism. Quite a number of configuration management systems take a similar approach – for example LCFG distributes configuration information via an XML representation [4], while bcfg2 configurations are specified in XML [5]. The rules described above *restrict* our XML representation to be simpler than would be allowed in plain XML, and allow it to be manipulated using a set of primitive operations (as described in the next section) which require the unique xpaths described above to address elements in the representation. It is those primitive operations to which authorisation rules are applied.

Configuration Manipulating Primitives

XML representations conforming to the rules described above may be manipulated using the following primitives:

- add_elt <element_path>
- del_elt <element_path>
- set_att <element_path> <attribute name> <value>
- del_att <element_path> <attribute_name>
- set_text <element_path> <text> NB – also erases any child elements.
- order <tag_path> <id> <first|last> or
  order <tag_path> <id1> [before|after] <id2>

All element_paths are *abbreviated xpaths* as described earlier. All primitives are *declarative* in the sense defined by LCFG [1], which means that add_elt and the like are slightly mis-named. A name more in keeping with it's function might be 'ensure_elt_exists', but this was judged a little unwieldy. It is also worthy of note that the primitives described above allow reasonably general XML to be constructed. This results in the above approach potentially having quite wide applicability.

In Machination these primitives are called *configuration instructions* and are collected together in a hierarchical structure as described later.

Authorisation of Configuration Instructions

The instruction set described above can be used to make demands about some configuration representation. They are very general – one can construct fairly arbitrary XML with them – and that they are reasonably small in number. It is to these primitives, or configuration instructions, that authorisation rules are applied.

Authorisation instructions are held in a database, and a typical instruction might be represented as

```
is_allow:    1
entities:    joe
operation:   add_elt
xml_path:    /profile/worker[packageman-1]
pattern:     <pattern>
             <constraint on="tag" type="string">package</constraint>
             <constraint on="id" type="set" set_id="unrestricted packages"/>
             </pattern>
```

Listing 1: Authorising configuration instructions.

something like that shown in Listing 1, which says that user joe[1] should be allowed to add sub-elements to /profile/ worker[packageman-1] as long as the sub-element's tag is "package" and its id matches the name of one of the elements of the set "unrestricted packages."

The schema for any eventual representation should be designed such that related pieces of configuration are bundled together in the representation. It is these related configuration pieces that form configuration *aspects*. An authorisation instruction relates to, and thus controls access to, an aspect if its xml_path attribute contains the XML path of the aspect.

The optional pattern clause allows the instruction to be more selective, subject to the content of the potential change. Tags and ids may be constrained for add_elt and del_elt, attribute names and values for set_att and so on. Current constraints allowed include string-wise equality, regular expressions, one of a provided list of strings, equal to the name (or other specified) attribute of one of the members of a set and their appropriate negatives.

Authorisation instructions control the ability of configuration instructions (primitives acting on XML) to access individual aspects of a configuration. The other promised topic for authorisation was authorising access to the structure which organises configuration information for lots of objects, which is introduced next.

[1]One can specify individual entities or sets of entities here, or a description using operators like or (require any of the pair), and (require both), and any N of some set.

Authorising Access to Aspects of Collections of Configurables

When configuring lots of objects, it is necessary to collect them together in some way. Strategies for this include database sets with no inheritance relationship (e.g., collections in Microsoft SMS), tree inheritance (e.g., OU tree in Microsoft Active Directory) and inverted tree inheritance (e.g., #include in LCFG, groups in Bcfg). SMS's unstructured sets lack an inheritance mechanism. The difference between the other two systems is similar to the difference between the *is-a* and *has-a* relationships in object oriented programming, though the distinction between the two is less clear in configuration management since 'inheritance' here is usually synonymous with accruing properties and values, and has no meaning in terms of method inheritance and method over-rides.

We now consider the implications of authorising actions on representations for these two types of inheritance.

is-a Inheritance

This kind of structure forms a tree of categories with the whole organisation at the root. The categories can be thought of as containers (as they are in Active Directory), containing configurable objects and other containers. Configuration instructions are attached to

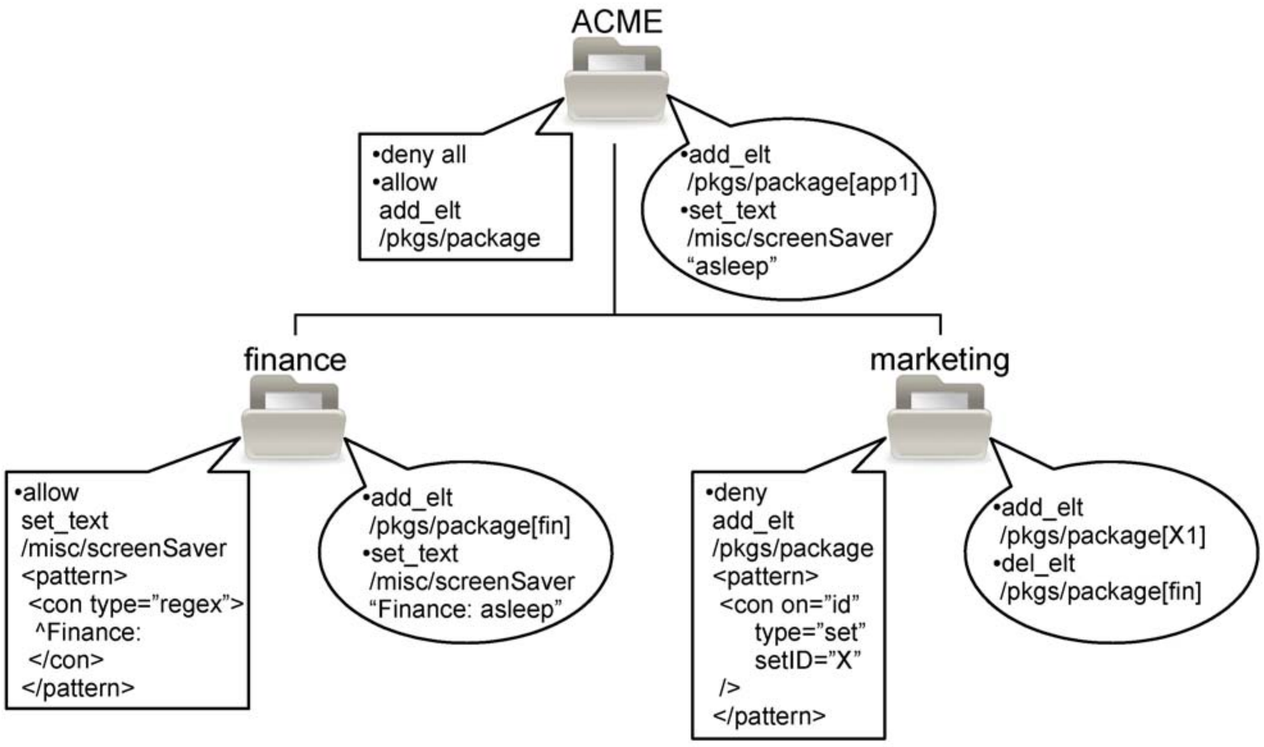

ACME
•deny all
•allow add_elt /pkgs/package
•add_elt /pkgs/package[app1]
•set_text /misc/screenSaver "asleep"
finance
marketing
•allow set_text /misc/screenSaver <pattern> <con type="regex"> ^Finance: </con> </pattern>
•add_elt /pkgs/package[fin]
•set_text /misc/screenSaver "Finance: asleep"
•deny add_elt /pkgs/package <pattern> <con on="id" type="set" setID="X" /> </pattern>
•add_elt /pkgs/package[X1]
•del_elt /pkgs/package[fin]

Figure 1: Is-a inheriting tree. Rectangular call-outs show authorisation instructions, round ones show configuration instructions. Containers inherit authorisation instructions from all containers above them in the tree. Some paths and instructions are abbreviated or omitted for reasons of space.

containers and apply to all configurable objects in that container or any of its sub containers.

Machination uses this style of inheritance (including multiple inheritance, as described below). Aspect authorisation instructions in Machination are attached to containers in the same way as configuration instructions, as shown in Figure 1. The authorisation instructions determine which configuration instructions are allowed to apply when the instructions are compiled into a representation, and are inherited from the root down the tree to the container being evaluated. A computer in the ''finance'' container in the tree shown would accrue instructions as shown in Table 1. Notice that the instruction to add package ''fin'' attached to the finance container is allowed because of an authorisation instruction attached to the ACME container.

has-a Inheritance

With this kind of structure, each configurable object has an associated tree rooted at that object. Each object can include or subscribe to collections of configuration instructions, and each such collection can include other collections.

For this style of inheritance aspect authorisation instructions could be included with the collections of configuration instructions as shown in Figure 2. Such authorisation instructions would determine which configuration instructions are allowed in the collection, which could be evaluated either at attachment time or at representation compile time.

The full list and order of authorisation instructions (and thus the results of the authorisation step) is order dependent, and cannot be determined from Figure 2 without further ordering information. This is explored in more detail later.

Multiple Inheritance

Either inheritance mechanism described above can support multiple inheritance. In is-a trees this is achieved by allowing objects to appear in more than one container and in has-a inverted trees by allowing

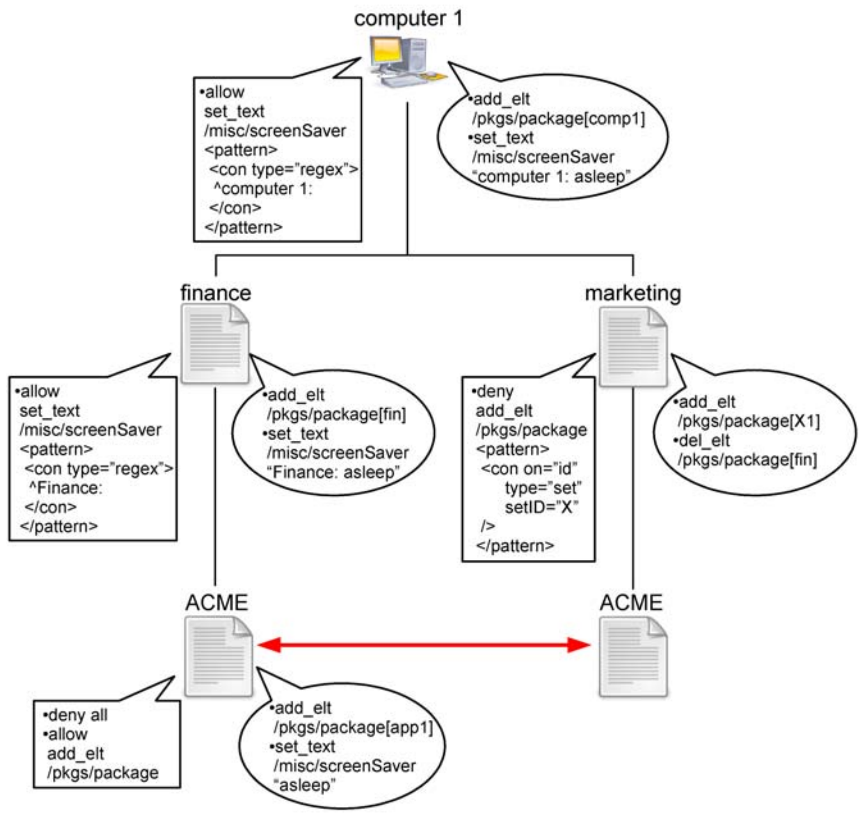


Figure 2: Has-a inheriting inverted tree. The two ACME icons represent the same file. Inheritance of authorisation instructions is order dependent, as explained later in the ''Conflicts and Merging'' section.

Source	Instruction	Allowed?	Reason
ACME	add package ''pkg1''	allowed	ACME allow packages
	set screensaver text ''asleep''	denied	ACME deny all
finance	add package ''fin''	allowed	ACME allow packages
	set screensaver text ''Finance: asleep''	allowed	finance allow screensaver

Table 1: Authorisation instructions for a ''finance'' computer.

more than one include or subscription. It was decided that Machination *should* support multiple inheritance to avoid the following two problems:

1. **Different tasks require different division criteria.** For example, consider the is-a tree in Figure 3. This would work well for configuration tasks that follow departmental boundaries, like "install finance_app1 on all finance computers," but would be rather clumsy for tasks like "make A4 paper the default on all computers in the UK," for which one would rather the computers were organised like the tree shown in Figure 4. A similar argument follows for delegation boundaries.
2. **Some objects fit more than one category.** In general, there will be objects that require configuration information for multiple reasons, and that therefore ought to be in multiple categories.

   As an example, consider the tree organised by department in Figure 3. Computers in the finance container have the finance suite of applications installed, while computers in the marketing container have the marketing suite. Now consider a computer that is used by both finance and marketing people, or by a person who has both finance and marketing roles. Such a computer needs both application suites (the union of the two sets of applications).

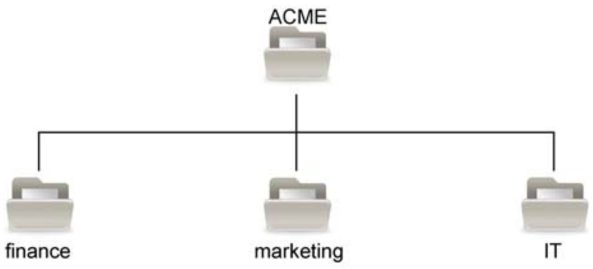


Figure 3: A simple tree showing ACME Corp. organised by department.

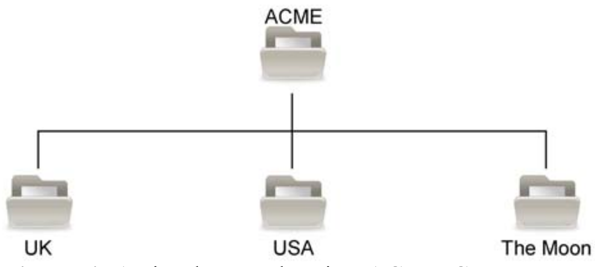


Figure 4: A simple tree showing ACME Corp. organised by location.

Conflicts and Merging

Supporting multiple inheritance immediately leads to the possibility of conflicts. Considering again the trees in Figures 1 and 2, if a computer exists in both the finance and marketing containers, or includes both finance and marketing files, it will have instructions both to add and remove the package "fin". Clearly these conflict. These configuration instructions must be merged in such a way that such conflicts are resolved.

The usual way this is resolved is to *order* the instructions. For is-a trees this means choosing a fixed evaluation order for all sibling containers. For example, the evaluation order of the finance and marketing containers in Figure 1 would need to be specified. For has-a inverted trees this means the inclusion order must be specified. In Figure 2, this would mean specifying the order in which the finance and marketing files were included, as well as whether the computer 1 instructions come before or after those inclusions and whether the ACME file is included before or after the instructions in each of the marketing and finance files.

Controlling the Hierarchy

So far the discussion has proceeded as if the hierarchy (is-a or has-a) is fixed, and the results are being computed based on placement of configurable objects and instructions within that hierarchy. The shape of the tree structure (includes or containers), the position of configuration and authorisation instructions, the position of configurable objects and the ordering of all of these are important both to the final configurations and to the nature of delegated portions. Changes to all of these need to be authorised.

It is beyond the scope of this paper to discuss authorisation of actions on the chosen hierarchy itself in any depth; however such operations fall more within the scope of the usual access control mechanisms to filesystem or directory objects. Machination's approach is to treat the hierarchy as a configurable object with a representation as defined earlier and to authorise modifications of that representation as described. Thus everything is unified under one authorisation system.

Choosing One

Both inheritance mechanisms described above have their advantages and disadvantages. The choice in Machination came from a trade-off between configuration flexibility and clarity for delegated contributors.

The has-a model is more flexible. The fact that instructions can be re-ordered on a configurable object by configurable object basis means that conflicting inclusions and instructions can be re-ordered to suit. However, such re-ordering of inclusions changes what delegated contributors are allowed to change, due to re-ordering of the associated authorisation information. This makes it less clear to contributors what they are and are not allowed to do. This and the possibility of interleaving includes, configuration instructions and authorisation instructions also make it more difficult to present such information to contributors in a graphical user interface.

The multiple inheriting is-a structure was chosen as the basis for the Machination hierarchy, in large part due to the relative ease of presenting compartmentalised views to delegated contributors.

Machination Specifics

The following features of the Machination hierarchy are more specific design choices in Machination and less relevant to the discussion on how to apply authorisation to general configuration systems. They are nonetheless important design choices with respect to the way that delegated portions of the Machination hierarchy interact.

Merge Policies

As mentioned earlier, choosing is-a style inheritance leaves Machination with a less flexible structure. We gain some flexibility back by introducing the concept of *merge policies* which may be attached to containers, and are applied when configuration instructions from multiple sibling containers are merged.

These specify the precedence of instructions between sibling containers depending on the contents of the instruction. Currently, the only information one can use is the element path to be altered, though this may be expanded to include details such as the attribute or text value being set. As an example, one could specify that the tree under the by network merge point should have precedence over the firewall area of a Machination Windows profile by attaching a merge policy of the form:

```
for xml_path /profile/worker[firewall-1] \
            local wins
```

where, similarly to the authorisation instruction shown earlier, the policy is stored as discrete data values in the hierarchy, rather than as a written command.

If there is a clash in merge policies (for example if two sibling containers' policies both claim precedence over one path in the profile) the siblings order in the parent is used to resolve precedence, just as if the policy statement did not exist.

Combining merge policies with authorisation instructions can be useful when delegating expertise specific configuration tasks. For example, suppose that an organisation has a networking expert. This expert should make sure that firewall rules are in place on each client appropriate for the client's network and status. We can set up a merge point called by network and delegate full control of the substructure to the expert (who will have a good idea of how to structure it for best results). The merge policy should be set to give precedence to that container for firewall rules, but to cede precedence for everything else and the authorisation instructions should allow configuration of only the firewall section of the profile. The expert can now organise computers into containers as required, perhaps resulting in a tree like that shown in Figure 5.

Merge Points

Multiple inheritance can be a dangerous thing. As a configurable object inherits configuration information from more paths it is more likely that conflicts will result and it becomes more difficult to follow and visualise a configuration back to its sources. Some temperance is required. As an aide to this (or perhaps an enforcement of it), Machination only allows configurable objects to be placed in two containers if they are separated by a *merge point*. For example, in Figure 5, a computer could be placed in both the low risk and subnet A containers, but not in both the low risk and high risk containers.

Merge points essentially break the multiply inheriting tree up into multiple singly inheriting trees, which are much simpler. Merge points should be inserted wherever one or both of the limitations of singly inheriting trees appears (i.e., different division

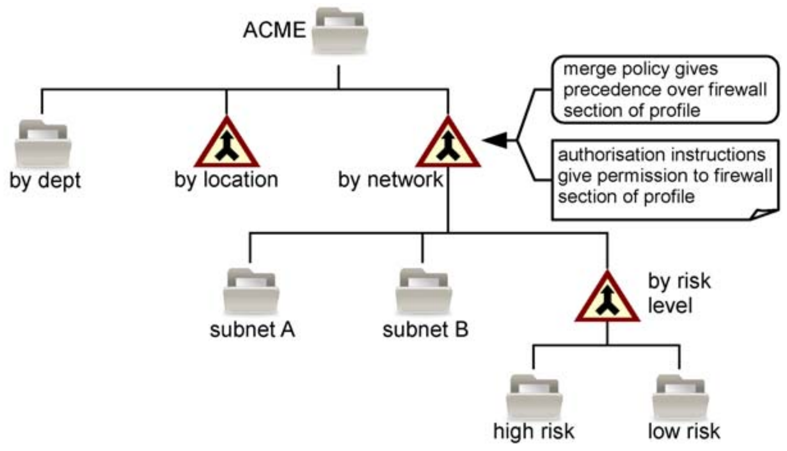

ACME
merge policy gives precedence over firewall section of profile
authorisation instructions give permission to firewall section of profile
by dept
by location
by network
subnet A
subnet B
by risk level
high risk
low risk

Figure 5: A networking expert has created a sub-tree in an area delegated for that purpose. The merge policy attached to "by network" gives this portion of the tree precedence over its siblings for firewall rules, which would otherwise normally take precedence due to the way they are ordered. Three containers ("by location", "by network", and "by risk level") have been nominated as merge points, which determine how configurable objects may be placed in the tree.

criteria or multiple categories are required). The idea is to use as many merge points as one needs, but as few as one can get away with.

Conclusions and Future Work

We have the beginnings of a system we hope will allow us to reduce our configuration workload at the same time as making our users and our managers happy. Our hope is based on experience with a useful but deficient system and underpinned by some new authorisation features. We are confident we will be able to delegate access to configuration aspects to desktop and laptop systems' end users, as well as more sophisticated forms of delegation with respect to collections of computers.

We believe that the authorisation work presented here, particularly the work on authorising access to individual configuration aspects, is transportable to other configuration systems and would be pleased to see others consider it. To this end, the Machination project is being open sourced, and code, or a link to it, should be available at the Machination project page [11] by the time this paper is published.

There is much work left to do, both on Machination and investigating more general consequences of authorising configurations. Some we have identified in the following.

- **Applicability to other configuration management systems** This paper describes a representation designed for configuration information and some primitives to build such representations which may be authorised. It goes some way toward describing how those configuration and authorisation instructions can be collected in the structures commonly used in other configuration systems before focusing on the structure used in Machination.
  From this follow two broad areas for possible investigation: could the given representation or another, better one be used to provide authorised access to configuration aspects across configuration management systems; and could the work on applying the authorisation instructions within other types of hierarchy be taken to the point where it is usable?
- **User interface** As stated in the section on goals, it is a requirement for us that the system be manipulated by a user interface which is suitable for use by a broad spectrum of users: possibly unskilled end users, computing professionals with expertise in some domain other than configuration management, and configuration experts with a good overview of the whole configuration system. Such an interface is currently under heavy development and not many details are available at this time, but the following are seen as requirements:
  - Unsophisticated use involving configuration of only one computer should be possible and easy. In most cases, this should involve picking things that the computer should "have" (like packages) from a list of available options. This mimics the current interface used to configure Machination computers.
  - Authorisation rules should be capable of allowing our common case of "you can configure the computer you are sitting at."
  - No one should be editing XML – not for configuration instructions, authorisation instructions or anything else. The XML is there for the computers to communicate with each other and a friendlier interface should be provided.
  - Sophisticated users should be able to view how given instructions will affect computers throughout the tree, how given computers inherit their configuration, where any conflicts are and a number of other types of information which span the whole or part of the tree. It is likely this information will be conveyed using toggled overlays.
  - Authorised contributors should easily be able to determine what they have control over.
- **Applicability to higher order configuration management** A matter of ongoing research in configuration management is how to raise the level of instruction from statements like "add package A" or (as a set of instructions) "install and configure a print service" to statements like "ensure there are two DHCP servers on every subnet" [9]. In a recent paper [8], Alva Couch also suggested that the semantic level of configuration management should be adjusted, such that configuration management systems reason more about the *meaning* of configuration instructions, rather than the eventually delivered content. This paper deals with authorisation at a more primitive level than either of these, and further investigation would be required to determine whether the authorisation schemes outlined in this paper could be relevant.
- **Building other representations** The representation rules outlined allow fairly general XML to be generated. The main restriction being the lack of mixed content elements. from this follow two areas of investigation: what things might one want to represent that cannot be represented without mixed content elements; and what other kinds of XML representation might a hierarchy like this usefully construct. On the later front, it has already been mentioned that Machination represents (and authorises actions on) its own state in this way. In fact the Machination hierarchy gives all configuration and authorisation instructions a service identifier,

which identifies which XML representation the instruction targets, and which can be used to keep separate several target representations.

Author Biography

Colin Higgs graduated from the University of Edinburgh with an Honours degree in Mathematical Physics. After a few years working as a physicist, Colin became the sole system administrator for the department of Chemical Engineering back at the University of Edniburgh. Several mergers and re-organisations later, he is now part of a larger team for the School of Engineering and Electronics, where he at least tries to work toward the "Bahamas" model of computing support.

Bibliography

[1] Anderson, Paul, *A Declarative Approach to the Specification of Large-Scale System Configurations*, 2001, http://www.dcs.ed.ac.uk/˜paul/publications/conflang.pdf .

[2] Anderson, Paul, George Beckett, Kostas Kavoussanakis, Guillaume Mecheneau, Jim Paterson, and Peter Toft, *Gridweaver Project Report D3.1: Large- Scale System Configuration with lcfg and smartfrog*, 2003, http://www.gridweaver.org/WP3/report3_1.
pdf .

[3] Anderson, Paul, and Alva Couch, *LISA 2004 invited talk: What is This Thing Called System Configuration?*, 2004, http://www.usenix.org/publications/library/proceedings/lisa04/tech/talks/ couch.pdf .

[4] Anderson, Paul and Alastair Scobie, "LCFG: The Next Generation," *UKUUG Winter Conference*, UKUUG, 2002, http://www.lcfg.org/doc/ukuug2002.pdf .

[5] *bcfg2 Documentation: Writing Specifications*, http://trac.mcs.anl.gov/projects/bcfg2/wiki/Writing Specification .

[6] *bcfg2 Home Page*, http://trac.mcs.anl.gov/projects/bcfg2 .

[7] *cfengine Home Page*, http://www.cfengine.org/ .

[8] Couch, Alva, "From x=1 to (setf x 1): What Does Configuration Management Mean?" *USENIX ;login:*, Vol. 33, Num. 1, pp. 12-18, 2008.

[9] Delaet, Thomas and Wouter Joosen, "Podim: A Language for High-Level Configuration Management," *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*, pp. 261-273, 2007.

[10] Iseminger, David, *Active Directory Services for Microsoft Windows 2000 Technical Reference*, pp. 28, 45, Microsoft Press, 2000.

[11] *Machination Home Page*, http://www.see.ed.ac.uk/machination .

[12] *Active Directory Home Page*, http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.mspx .

[13] *Microsoft Systems Management Server Home Page*, http://www.microsoft.com/smserver/default.mspx .

[14] *Reductive Labs Puppet Project Page*, http://reductivelabs.com/projects/puppet/ .

[15] *LCFG Home Page*, http://www.lcfg.org .

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of system administrators, developers, programmers, and engineers working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering technical excellence and innovation
- encouraging computing outreach in the community at large
- providing a neutral forum for the discussion of critical issues

For more information about membership and its benefits, conferences, or publications, see http://www.usenix.org.

SAGE, a USENIX Special Interest Group

SAGE is a Special Interest Group of the USENIX Association. Its goal is to serve the system administration community by:

- Establishing standards of professional excellence and recognizing those who attain them
- Promoting activities that advance the state of the art or the community
- Providing tools, information, and services to assist system administrators and their organizations
- Offering conferences and training to enhance the technical and managerial capabilities of members of the profession

Find out more about SAGE at http://www.sage.org.

Thanks to USENIX & SAGE Corporate Supporters

USENIX Patrons

Google™ Microsoft® Research NetApp™

USENIX Benefactors

hp invent IBM LINUX PRO MAGAZINE vmware®

USENIX & SAGE Partners

Ajava Systems, Inc.
DigiCert® SSL Certification
FOTO SEARCH Stock Footage and Stock Photography
Splunk
Zenoss

USENIX Partners

Cambridge Computer Services, Inc.
GroundWork Open Source Solutions
Hyperic
Infosys
Intel
Oracle
Sendmail, Inc.
Sun Microsystems, Inc.
Xirrus

SAGE Partner

MSB Associates

ISBN-13: 978-1-931971-63-8
90000
9 781931 971638